OK2440 用户手册 V1.0

保定飞凌嵌入式技术有限公司

网站：http://www.witech.com.cn

论坛：http://www.witech.com.cn/bbs

目 录

一　预装 Wince 使用说明

WINDOWS CE 是微软开发的一个开放的、可升级的 32 位嵌入式操作系统，是基于掌上型电脑类的电子设备操作系统。它是精简的 Windows 95。Windows CE 的图形用户界面相当出色，其中 CE 中的 C 代表袖珍(Compact)、消费(Consumer)、通信能力(Connectivit)和伴侣(Companion);E 代表电子产品(Electronics)。与 Windows 95／98、Windows NT 不同的是，Windows CE 是所有源代码全部由微软自行开发的嵌入式新型操作系统，其操作界面虽来源于 Windows 95／98，但 Windows CE 是基于 Win32 API 重新开发的、新型的信息设备平台。Windows CE 具有模块化、结构化和基干 Win32 应用程序接口以及与处理器无关等特点。Windows CE 不仅继承了传统的 Windows 图形界面，并且在 Windows CE 平台上可以使用 Windows 95／98 上的编程工具（如 Visual Basic、Visual++等）、使用同样的函数、使用同样的界面风格，使绝大多数的应用软件只需简单的修改和移植就可以在 WindowsCE 平台上继续使用。

Windows CE 的设计目标是：模块化及可伸缩性、实时性能好，通信能力强大，支持多种 CPU。它的设计可以满足多种设备的需要，这些设备包括了工业控制器、通信集线器以及销售终端之类的企业设备，还有像照相机、电话和家用娱乐器材之类的消费产品。一个典型的基于 Windows CE 的嵌入式系统通常为某个特定用途而设计，并在不联机的情况下工作。它要求所使用的操作系统体积较小，内建有对中断响应功能。

WINDOWS CE 的特点有：

(1)具有灵活的电源管理功能，包括睡眠／唤醒摸式。

(2)使用了对象存储（object store）技术，包括文件系统、注册表及数据库。它还具有很多高性能。高效率的操作系统特性，包括按需换页、共享存储、交*处理同步、支持大容量堆（heaP）等。

(3)拥有良好的通信能力。广泛支持各种通信硬件，亦支持直接的局域网连接以及拨号连接，并提供与 PC、内部网以及 Internet 的连接，还提供与 Windows gx／NT 的最佳集成和通信。

(4)支持嵌套中断。允许更高优先级别的中断首先得到响应，而不是等待低级别的 ISR 完成。这使得该操作系统具有嵌入式操作系统所要求的实时性。

(5)更好的线程响应能力。对高级别 IST（中断服务线程）的响应时间上限的要求更加严格，在线程响应能力方面的改进，帮助开发人员掌握线程转换的具体时间，并通过增强的监控能力和对硬件的控制能力帮助他们创建新的嵌入式应用程序。

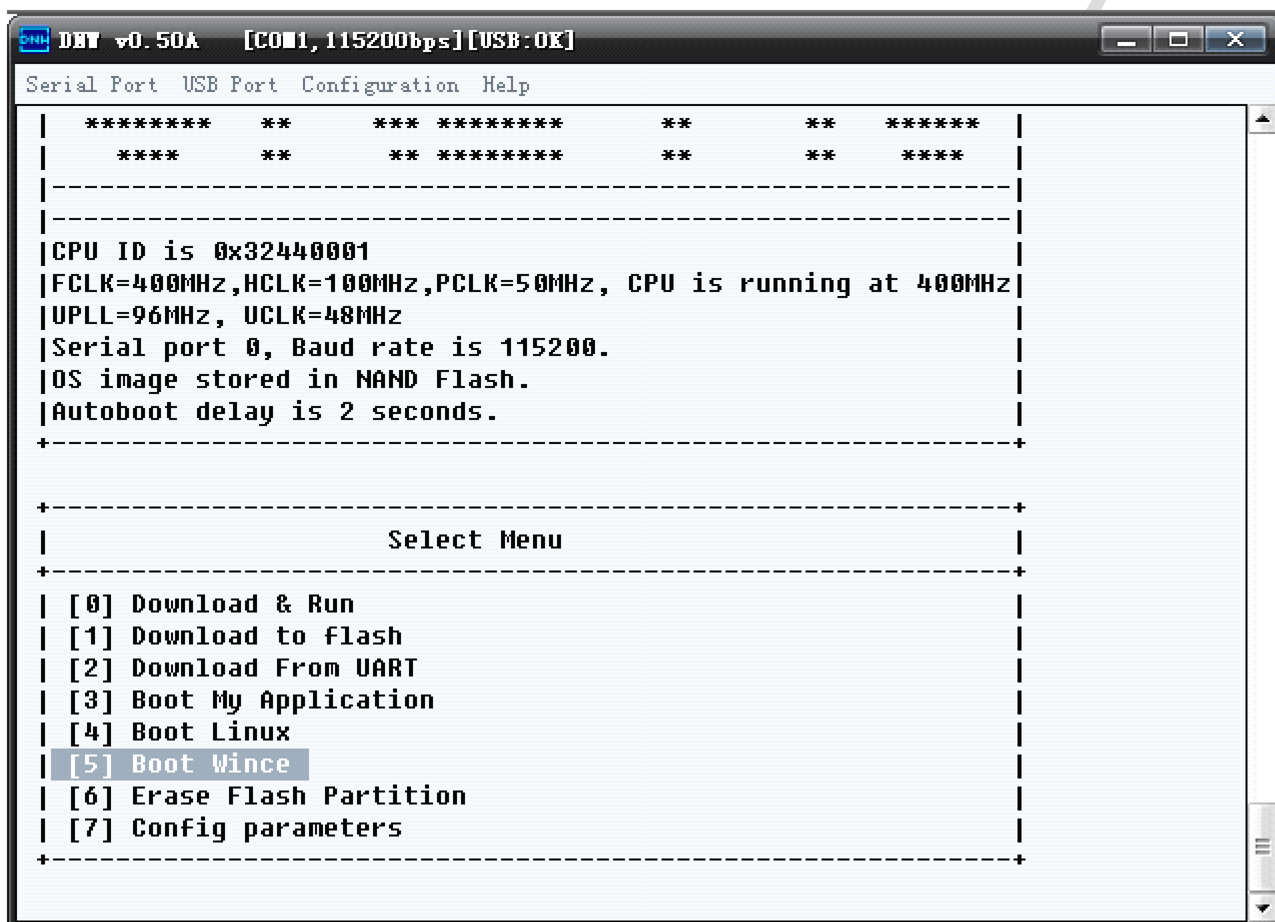
(6)256 个优先级别。可以使开发人员在控制嵌人式系统的时序安排方面有更大的灵活性。

(7)Windows CE 的 API 是 Win32 API 的一个子集，支持近 1500 个 Win32 API。有了这些 API，足可以编写任何复杂的应用程序。当然，在 Windows CE 系统中，所提供的 API 也可以随具体应用的需求而定。

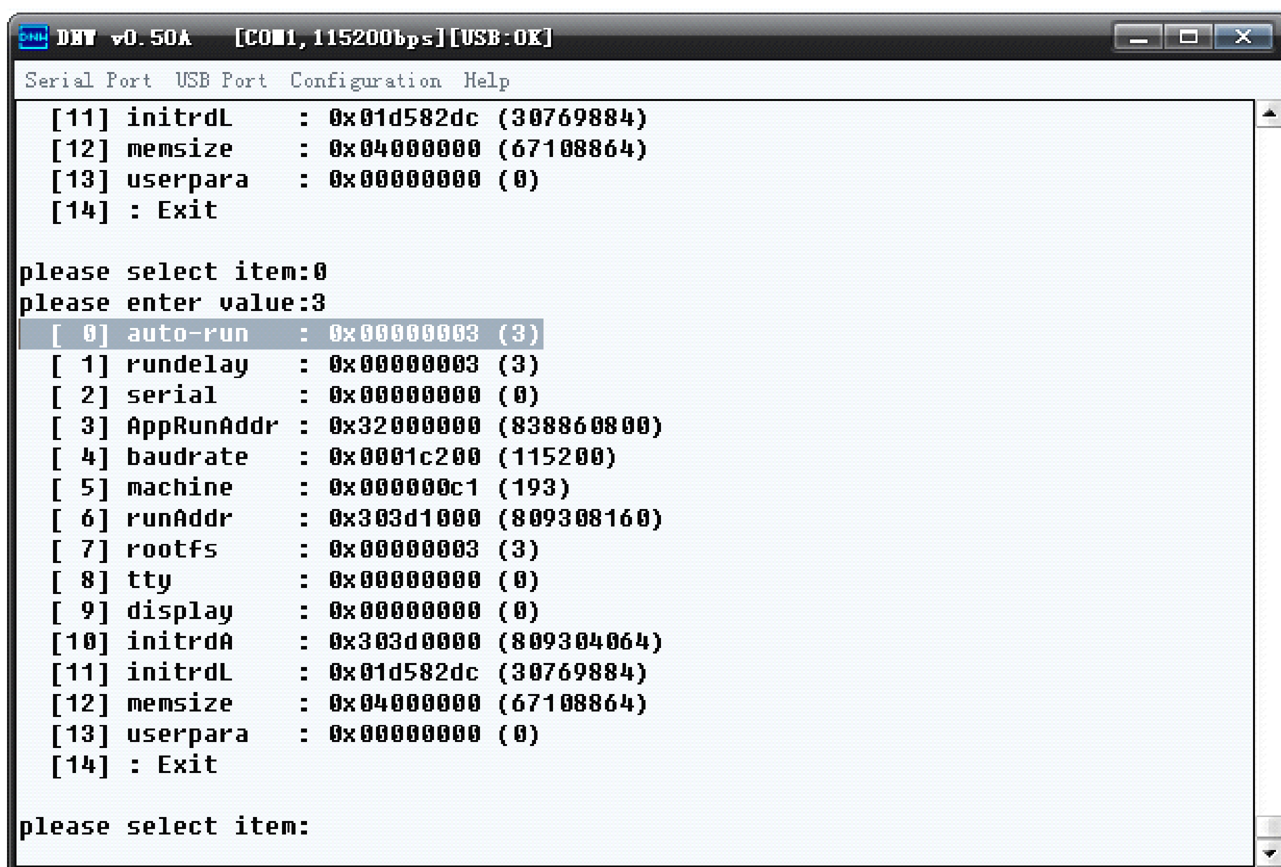
在掌上型电脑中 Windows CE 包含如下一些重要组件：Pocket Outlook 及其组件、语音录音机、移动频道、远程拨号访问、世界时钟、计算器、多种输入法、GBK 字符集、中文 TTF 字库、英汉双向词典、袖珍测览器、电子邮件、Pocket Office、系统设置、Windows CE Services 软件。

1.1 如何启动 WINCE

OK2440 开发板出场时已经预装了 wince 操作系统，在 bootloader 菜单中选择 5 来启动 wince。

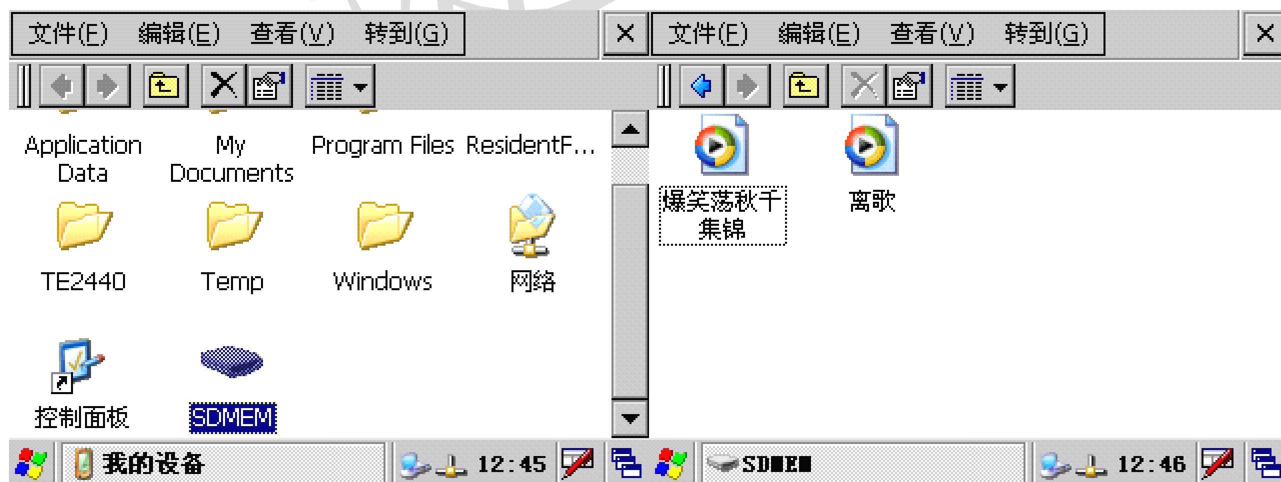


也可以通过修改 Bootloader 参数实现 Wince 的上电自启动，将参数 0 的值改为 3 即可(2 为自启动 Linux)，另外可通过修改参数 1 的值来改变启动延时：



1.2 如何使用 SD 卡

启动前插上 SD 卡，wince 启动后可在我的设备中看到 SDMEM



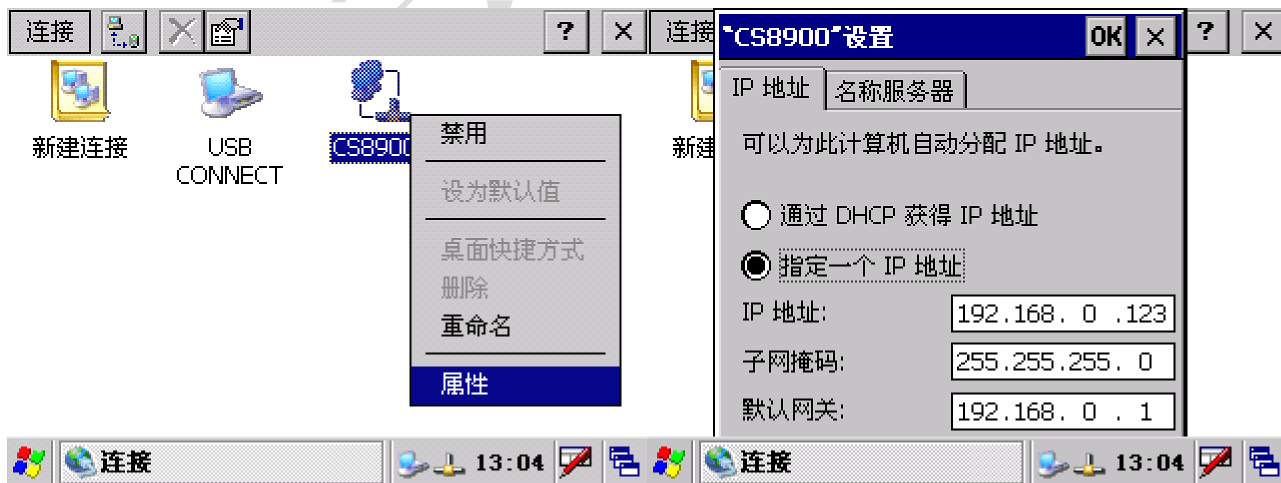
1.3 使用 Windows Media Player 播放 MP3 和视频

出厂时预装的 wince 操作系统带有 Media Playe 播放器，直接双击要播放的音乐文件或视频文件即可播放。



1.4 如何浏览网页

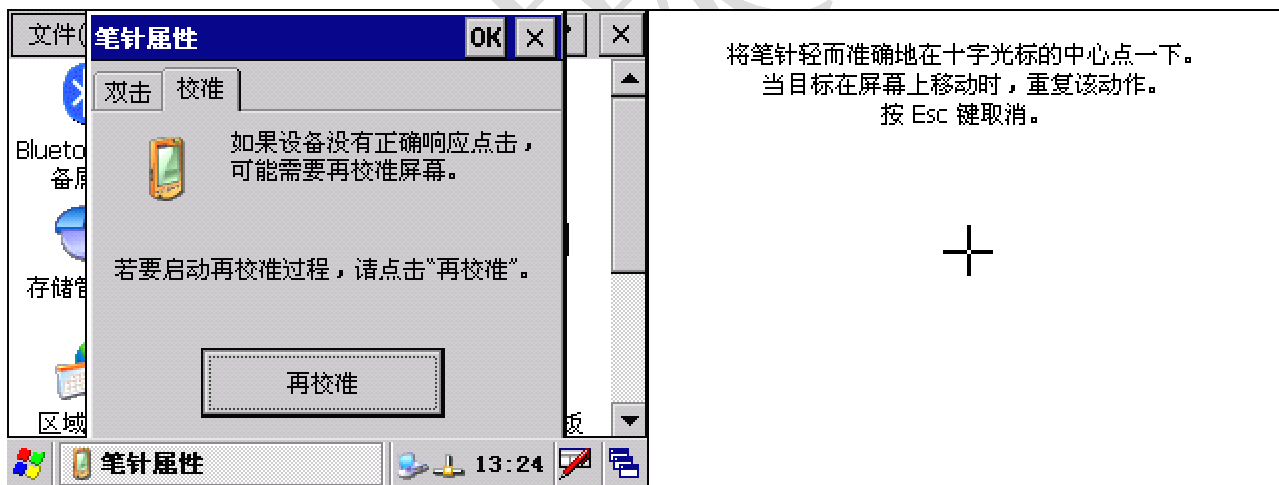
wince 启动后点击我的设备—>控制面板—>网络和拨号连接，在 CS89001 上点右键—>属性，弹出网络设置对话框进行网络设置，然后就可以用 IE 浏览网页了。





1.5 触摸屏校正

第一次使用触摸屏的时候需要先校准一下，因为预装的 wince 操作系统添加了保存注册表功能，所以校准一次后下次就不用再校准了。打开我的设备—>控制面板—>笔针，点击校准菜单上的‘再校准’按钮开始校准触摸屏。

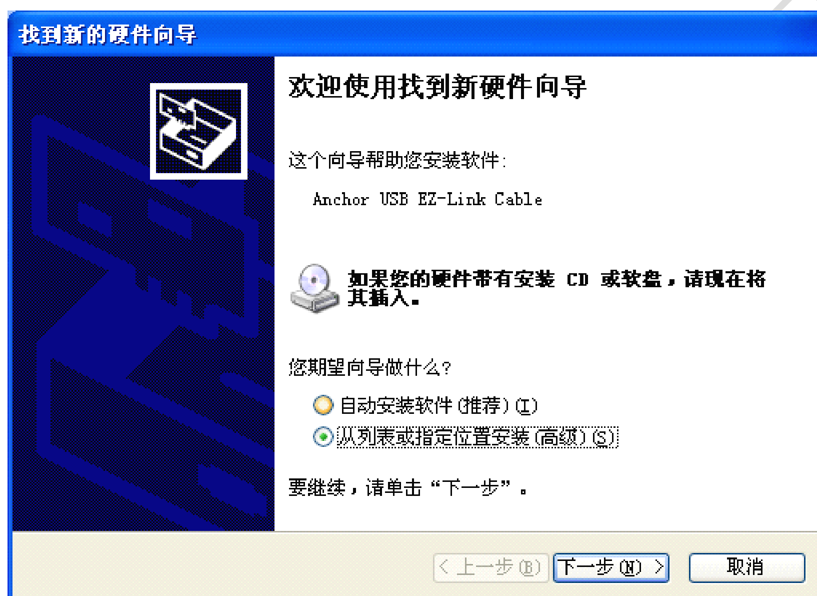


1.6 在 WINCE 和桌面系统之间建立连接

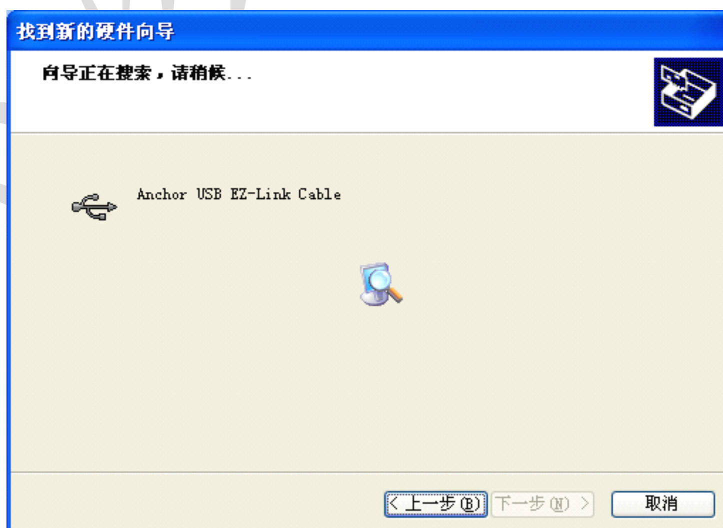
可以使用微软公司提供的一些工具，并在进行了一些必要的设置后，就可以在安装了 WINCE 操作系统的移动设备和 windows 桌面系统之间进行通讯连接，从而可以实现文件上传下载，远程调试等功能。

1.6.1 安装驱动

启动 WINCE 后，用 USB 线连好 USBDEIVCE 和 PC 的 USB 端口，如果以前没有安装 WINCE 下的驱动，这时插上 USB 线后，在计算机端会出现“发现新硬件”的提示，这时就需要安装驱动了，驱动的位置在 wince 的 BSP 包里（WINCE500/platform/ok2440/DRIVERS/USB/FUNCTION），安装好 USB 驱动，就可以进行下面的操作了。



选择驱动所在的位置。

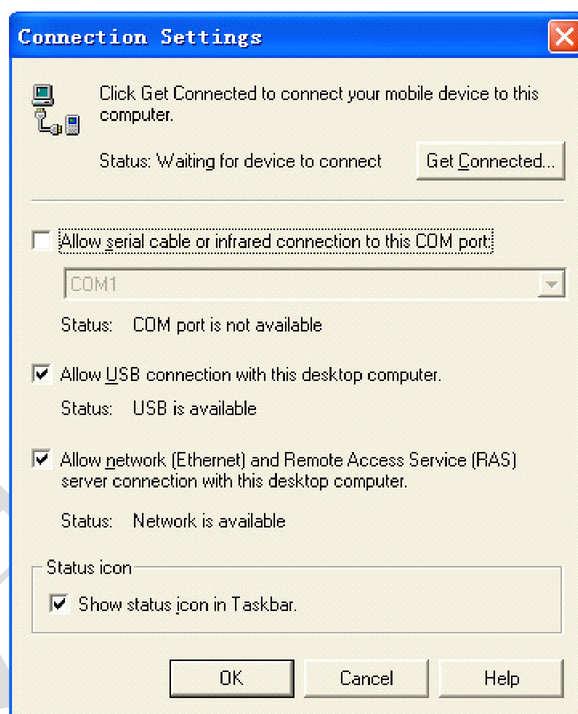


注意：该驱动和 2.1.1 章节所说的驱动是不一样的。需要首先安装前者，wince 成功启动后再安装后者。

1.6.2 使用微软 ActiveSync 同步传输工具进行通讯连接

首先下载 ActiveSync 工具的安装程序 MSASYNC.EXE（这个工具可以单独从网上下载），安装，运行。

然后开始对 ActiveSync 进行设置，点击菜单 File | Connection Settings ，在弹 出的设置对话框中，选择允许通过 USB 口、以太网口进行通讯连接，串口就不用选上了，以免同 DNW 冲突。如下图所示：

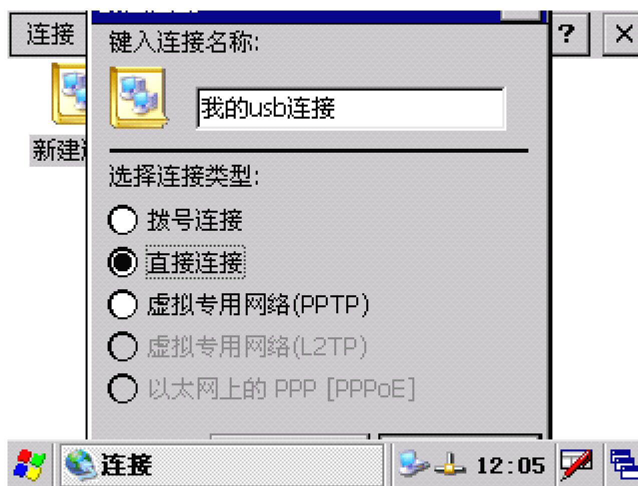


ActiveSync 可以通过串口，USB 口，网口等方式建立连接，下面仅以 USB 连接方式为例介绍如何通过 ActiveSync 在桌面系统与 WINCE 之间建立通讯连接。

一般情况下 WINCE5.0 自动配置了 USB 连接方式，用户只需重新插拔一下 USB 线即可，也可按以下步骤进行重新配置。(之前应校准系统时间，否则可能会连接不上）

第一步，在确认桌面系统的 USB 口和 OK2440 的方形 USB 口已经通过 USB 线缆连接起来后，开始按以下步骤设置 OK2440 上运行的 wince 操作系统： 打开【我的电脑】—》打开【控制面板】—》打开【网络和拨号连接】—》点击【新建连接】，在“新建连接”设置对话框中，选择连接类型为“直接连接”，

如下图:

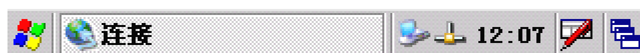


改名输入“我的 usb 连接”，点击“下一步”，在出现的“选择设备”下拉列表中选择“S3C2440 USB Cable:”，如下图:

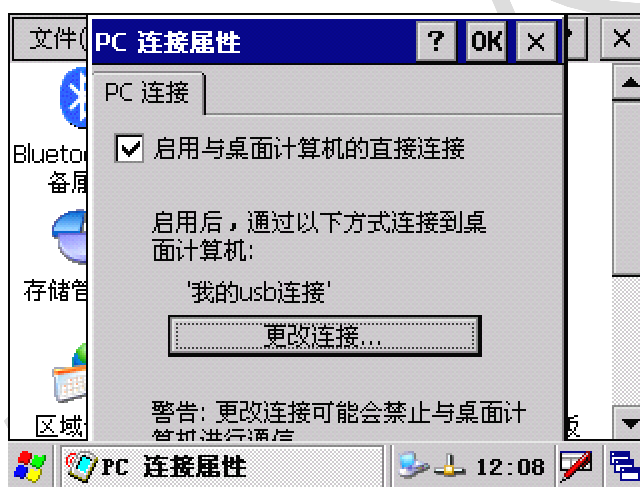


点击完成，这时就出现“我的 USB 连接”图标，如下图。

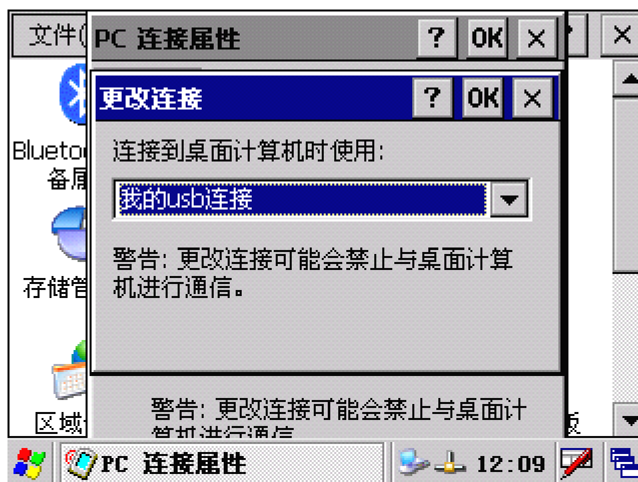




回到“控制面板”，点击【PC 连接】图标，进入“PC 连接属性”设置对话框，选中“启用与桌面计算机的直接连接”的复选框，然后再点击“更改连接”按钮， 如下图，



在“更改连接”设置对话框的下拉列表框中选择刚才新建的连接“我的 USB 连接"，然后按“OK”键退出。这样 WINCE 的设置便完成了。



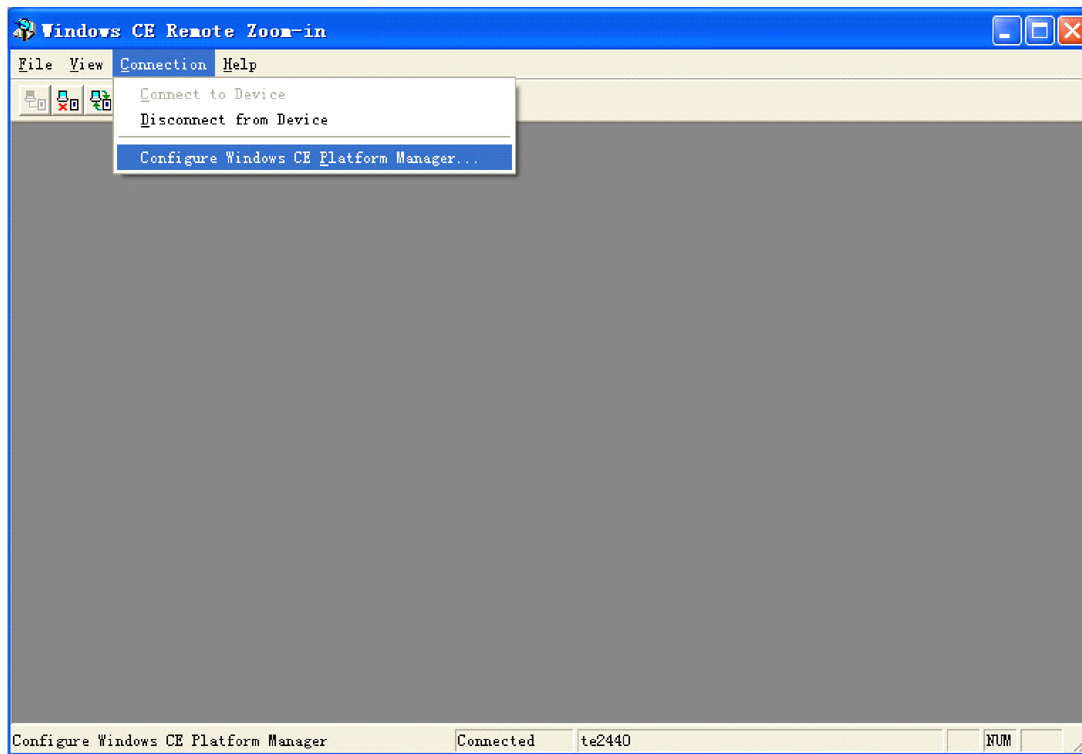
最后，将 USB 连接线重新插拔一次，就开始自动连接了。连接成功后，ActiveSync 的图标会变成另外一种颜色，并且提示连接成功。这时，打开菜单 File | Explore，就可以浏览 WINCE 系统上的资源，也可以通过复制/粘贴的方式在系统之间拷贝文件。



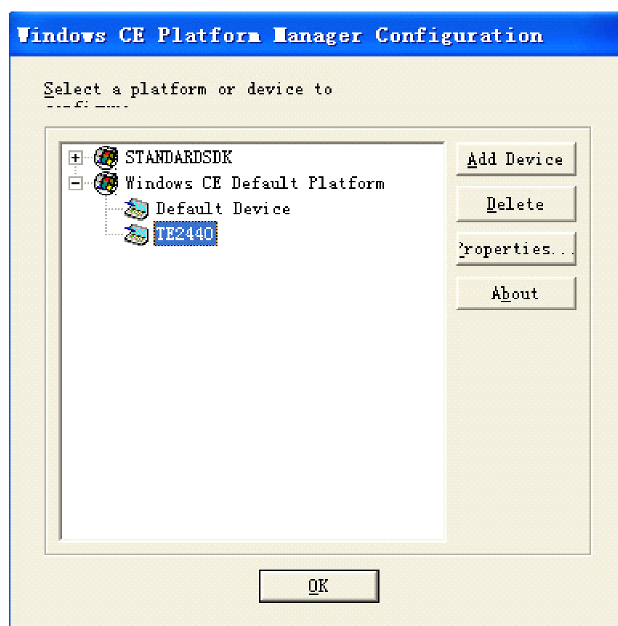
1.7 如何为 wince 屏幕抓图

ActiveSync 的成功连接是使用所有微软远程连接工具的基础。在 ActiveSync 成功连接后，就可以进

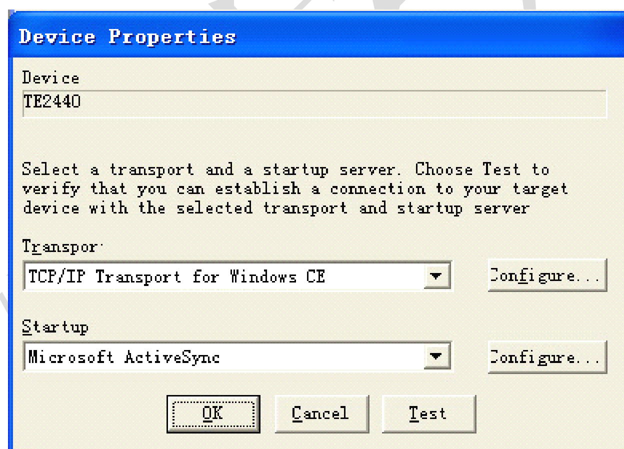
一步使用到 PB 程序的“Tools”菜单项里面的许多远程工具。例如点击 PB 的 Tools | Remote zoom-in 菜单，以运行远程图片缩放工具，这个程序可以实现对远程移动设备显示屏幕的截屏。Remote zoom-in 程序运行起来后，首先要配置一下平台管理器，点击它的 Connection | Configure windows ce platform manager 菜单，如下图：



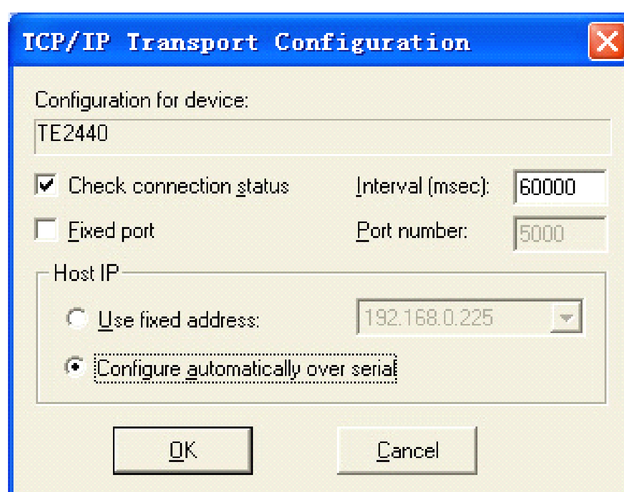
在弹出的对话框中添加一个新设备，设备名可以取为“OK2440”，如下图



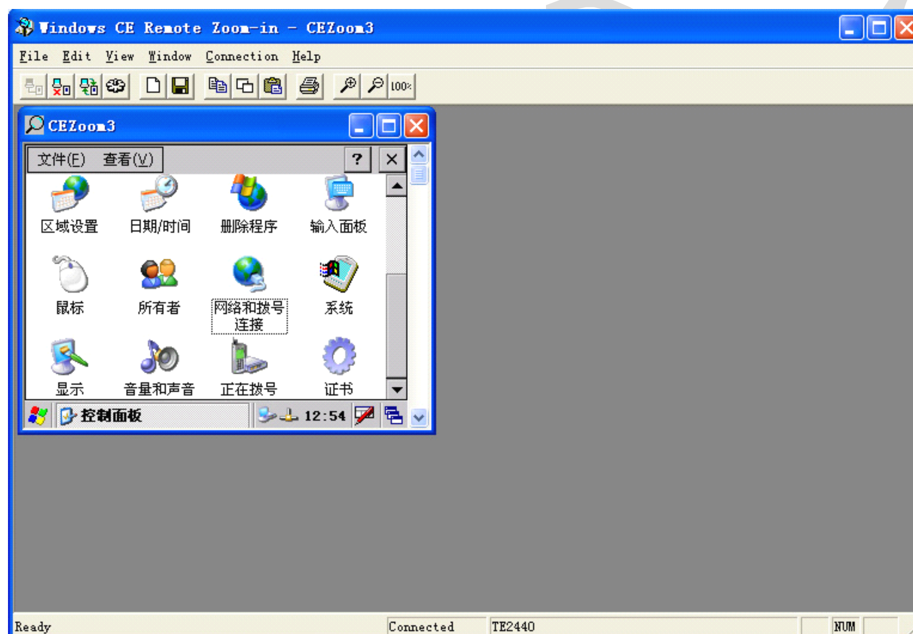
点击右边的“Propertions…”,设置“OK2440”设备的属性，如下图



点击“Transport”下拉框右边的“Configure”按钮，以设置 TCP/TP 传输的设置，按下图所示进行配置。



设置结束后，点击 connect 按钮进行连接，连接成功后便可以截图了。



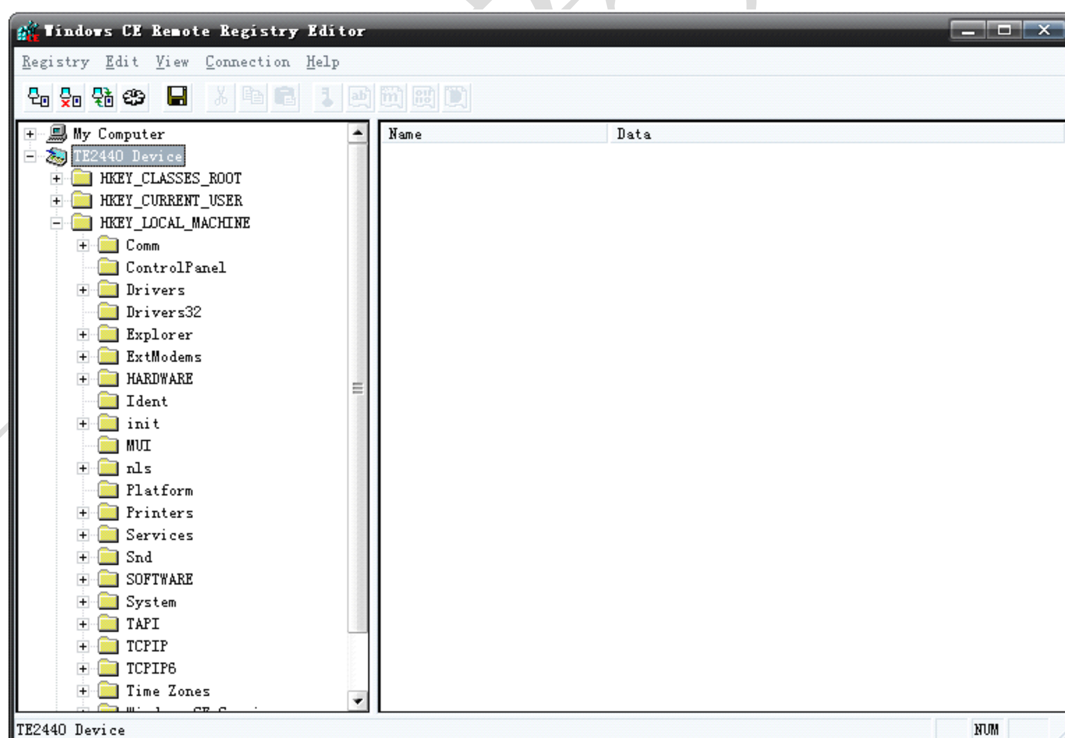
1.8 查看修改 wince 注册表

在 ActiveSync 成功连接后，点击 PB 的 Tools | Remote Registry Editor 菜单，以运行远程注册表管理程序，这个程序可对远程移动设备的注册表进行编辑。

首先在弹出的选择设备对话框中选择我们上一节新建的设备，点击‘OK’。



接下来就可以查看或编辑 wince 的注册表了



注：在没有成功连接到网络设备时，窗体左侧只有一个 My Computer，这个是本机的注册表。

1.9 如何动态修改液晶分辨率

目前市场上的支持 wince 系统的开发板，绝大部分只支持单一分辨率，也就是说他可能只支持 320*240 的液晶屏，你换个高分辨率的大屏也是白搭；如果它只支持 640*480 的大屏，而此时你手中只有 320*480 的小屏，这时候你就麻烦了，在买块大屏是你的首选，否则你只能看到桌面一角了，所以初学者一定要注意。针对这个问题，OK2440 为您解决后顾之忧。该开发板同时支持两种分辨率 320*240 和 640*480，操作步骤如下：

1、将液晶与板子接好，然后上电启动 wince。如果板子自启动是 wince 还好，如果不是则可以在 bootloader 界面选择 5 来启动 wince。当起来后，液晶会显示大家都很熟悉的桌面，类似 windows。

2、这时候你会在发现桌面上有个 ScreenControl 的快捷方式，双击它会弹出相应的对话 ，这时你可以用鼠标或是触摸笔来修改分辨率，选好后按 OK。然后重新上电或复位后你就可以使用您设好的分率的液晶屏了。作为开发板的技术支持，强烈建议您如果没有必要不要随便修改分辨率。因为您如果使用的是低分辨率的 LCD,而不小心改了高分辨率，这时候只有换块大屏或是重装内核才能修改回来

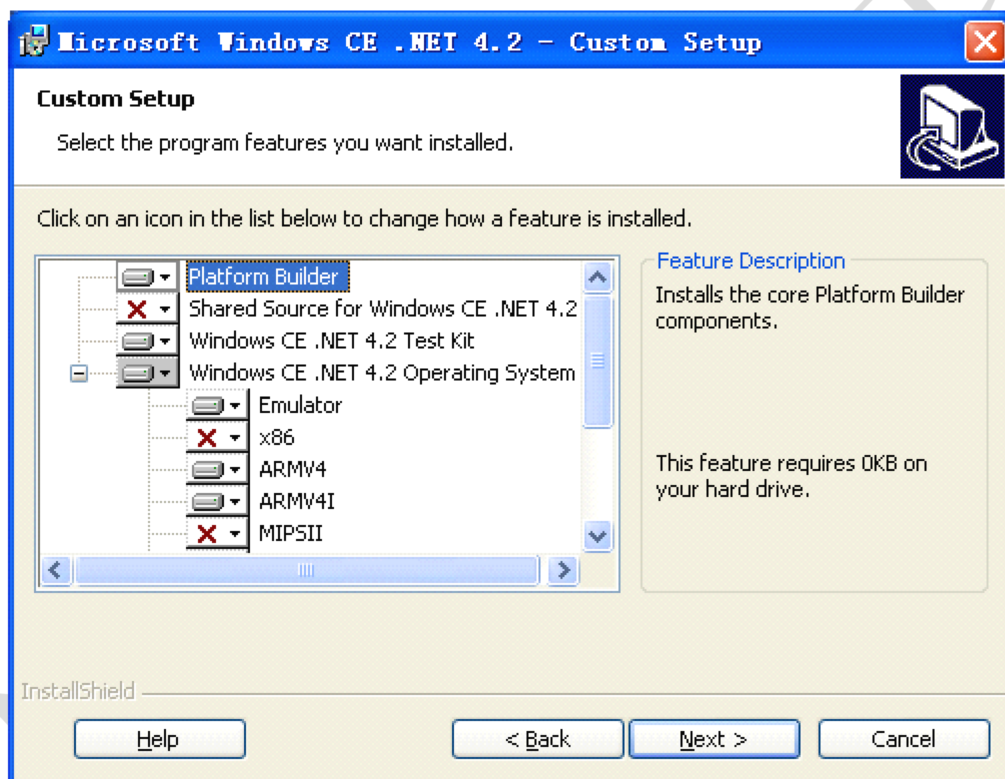
1.10 应用程序的运行

如果用 EVC 或 EVB 编译了自己编写的 WINCE 应用程序，并点击在“tools”菜单的“Configure platform manager”菜单项完成配置平台管理器后，点击运行按钮，就可以自动的把程序发送到板子上运行，此外也可以利用 sync 工具的 explorer 将编译好的程序上传到 OK2440 上运行。

二　Wince 开发教程

2.1 安装 Windows CE.NET 开发环境

为了建立 WINCE5.0 的应用开发环境，您需要准备好由微软公司发布的 PB（PlatformBuilder5.0）安装光盘，这张光盘包含了微软公司的 Windows®CE.NET5.0 操作系统安装程序，以及把操作系统编译移植到指定目标硬件平台的工具——平台建立器（PlatformBuilder5.0）。下面介绍如何安装 WINCE5.0 操作系统和 PlatformBuilder5.0。首先运行 SETUP.EXE,输入串号。在出现安装选项时选择 ARMV4，ARMV4I，如图：



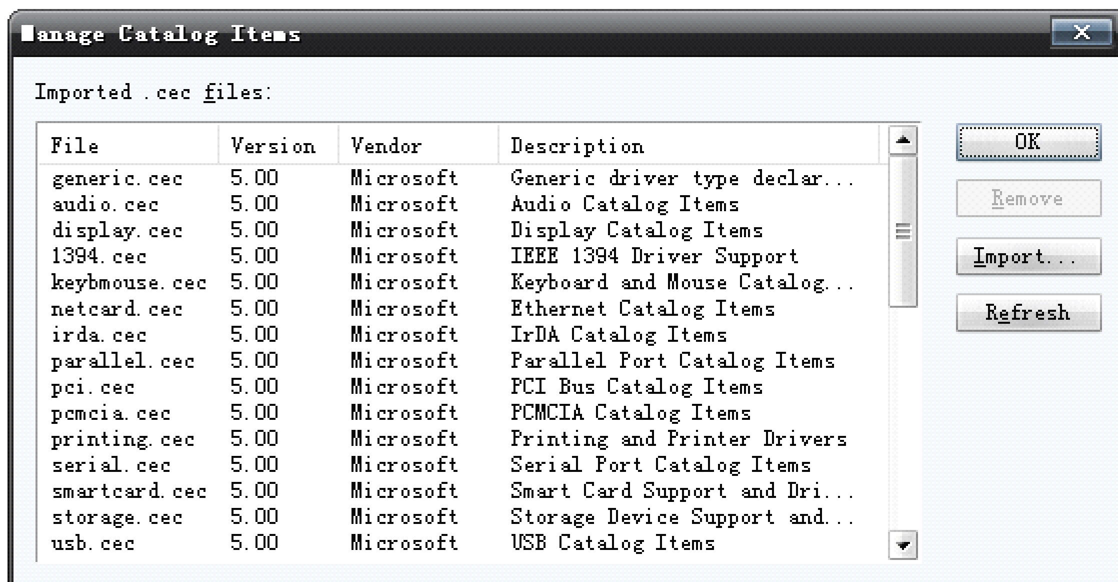
一直点 next 安装结束。WINCE 目录有 3.5G 左右的文件。

2.2 安装基于 OK2440 的 BSP 包

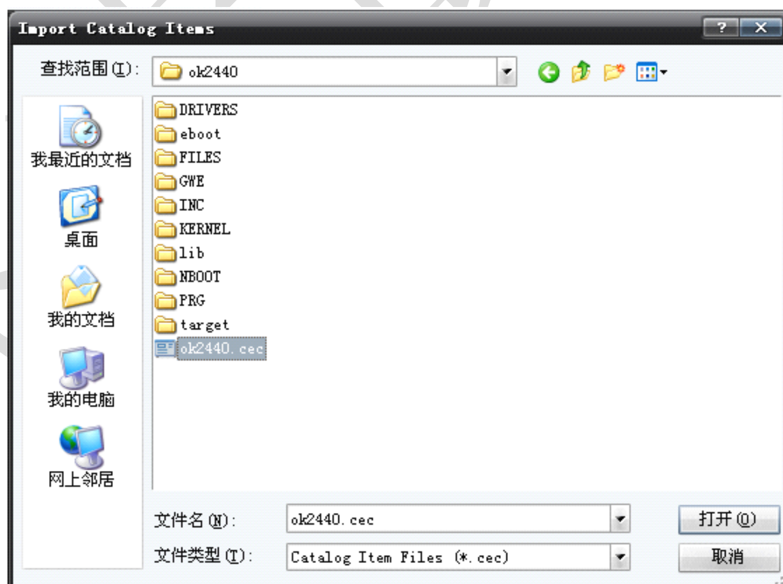
前面已经安装了平台建立器 Platform Builder 5.0，下一步，要运行 Platform Builder，并对它进行一系列的设置，目的是为编译 WINCE5.0操作系统映像做好准备。先把光盘中的 WINCE 的 BSP 包 OK2440.rar 拷贝到 WINCE500 的安装路径下的 PLATFORM 文件夹中去（笔者这里为 E:\WINCE500\PLATFORM\），并解压，

解压后在此目录下出现一个新的文件夹 OK2440（E:\WINCE500\PLATFORM\OK2440），WINCE500 的 BSP 工作目录源码就在此路径下面。

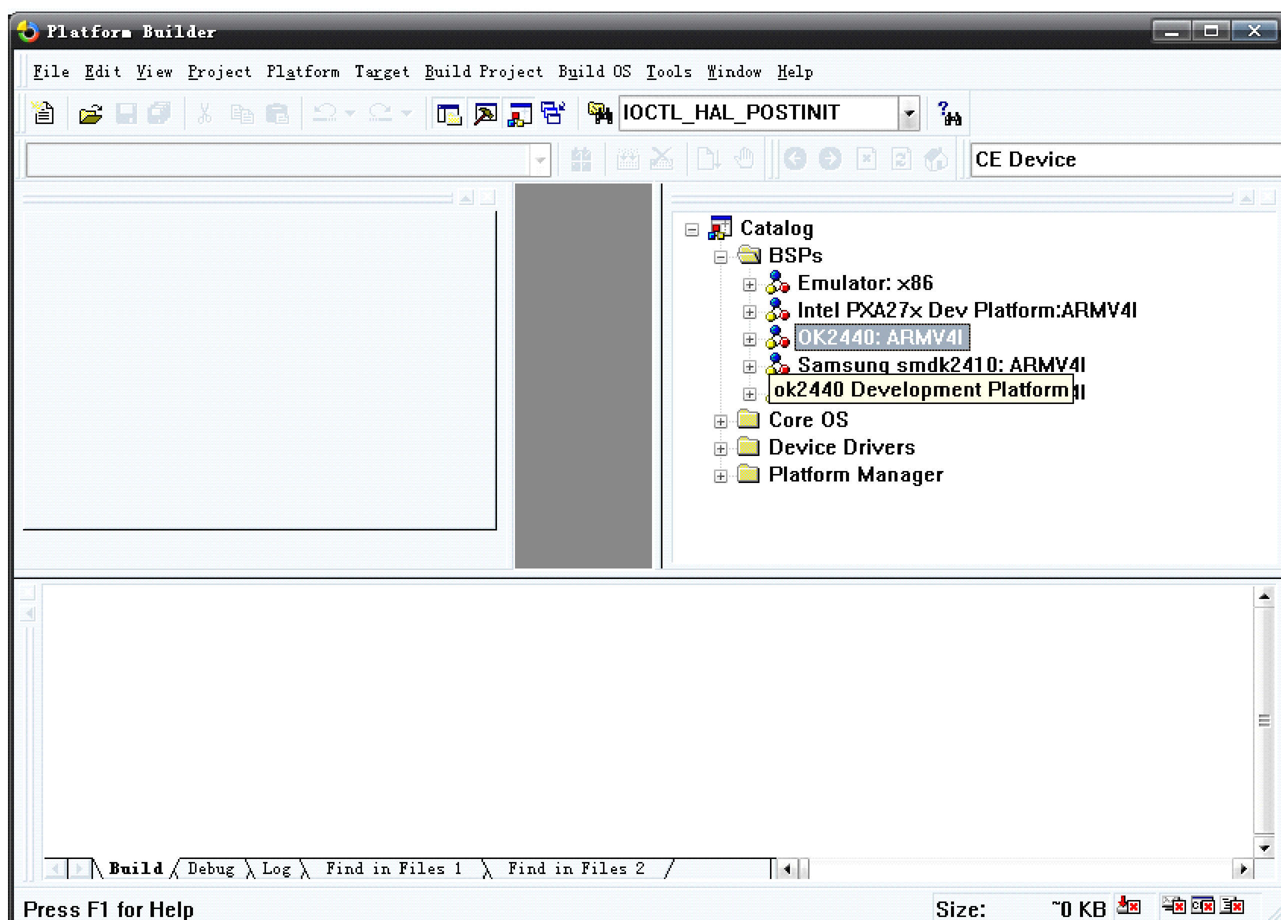
打开 Platform Builder 5.0 编译工具，点击 PB 主菜单“File”下的“Manage Catalog Features”菜单项，弹出 Catalog 管理窗口：



点击“Import” 按钮， 打开 WINCE420\PLATFORM\OK2440 目录，选中该目录下的“OK2440.CEC”文件，点打开按钮导入该文件：



导入之后，在 PB 的目录查看器上将会看到“OK2440:ARMV4I”列，如下图：



2.3 更新开发环境（PB5）

到目前为之，官方一直在不断的更新 wince5.0 开发环境，我们可以到官方网站上下载补丁包进行更新。网址为：

http://www.microsoft.com/downloads/results.aspx?pocId=&freetext=Windows%20CE%205.0%20Platform%20Builder%20Update&DisplayLang=en

我们在用户光盘上提供了 2007 年一月份以前的所有更新：

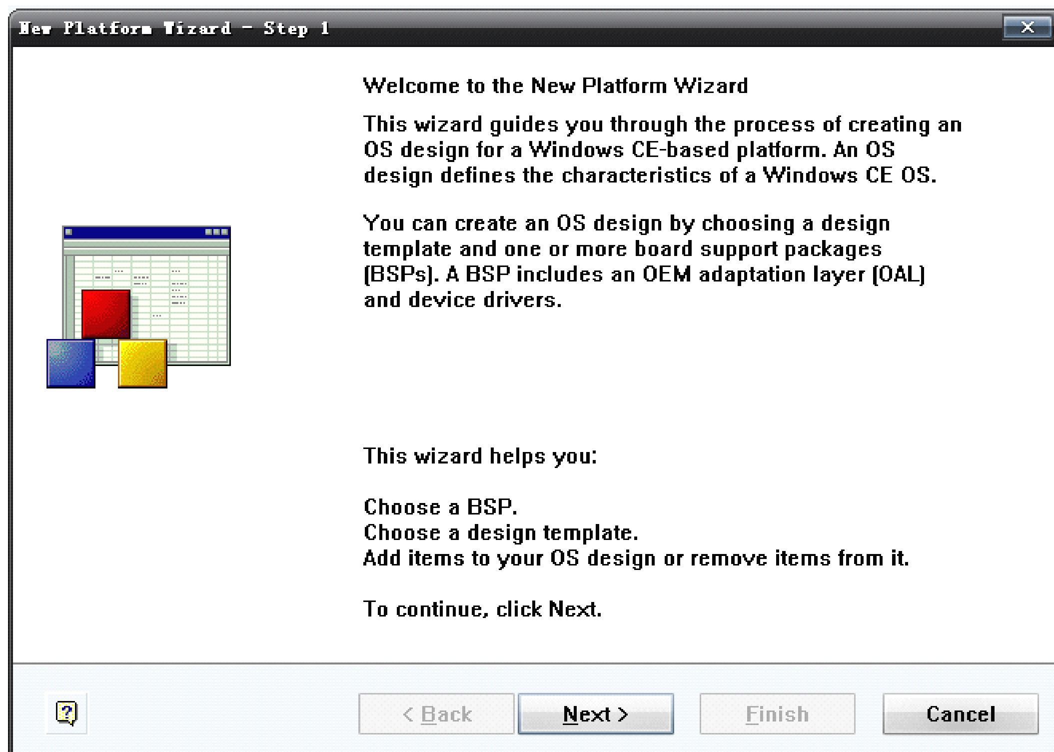
OK2440\wince\WinCEPB50-061231-Product-Update-Rollup-Armv4I.msi，双击安装即可。

注：若要 wince 支持.net2.0，必须安装该更新程序。

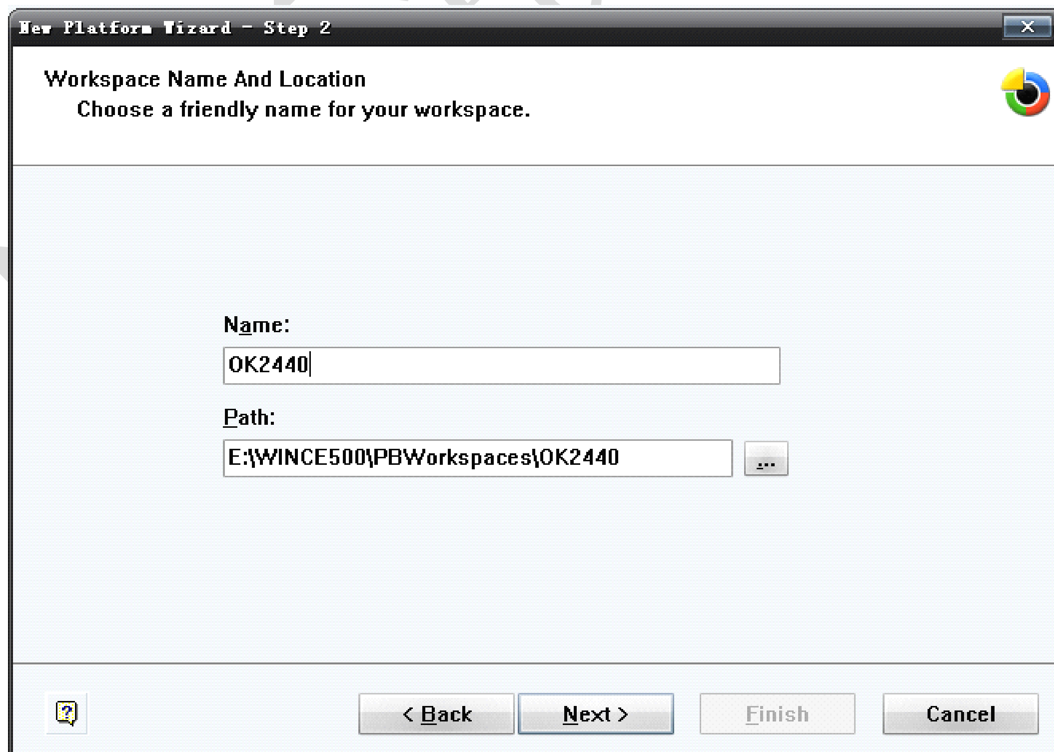
2.4 新建工程

点击 PB 主菜单“File”下的“New Platform”菜单项，将会出现“New Platform Wizard - Step 1”

框，点击“Next” 按钮：

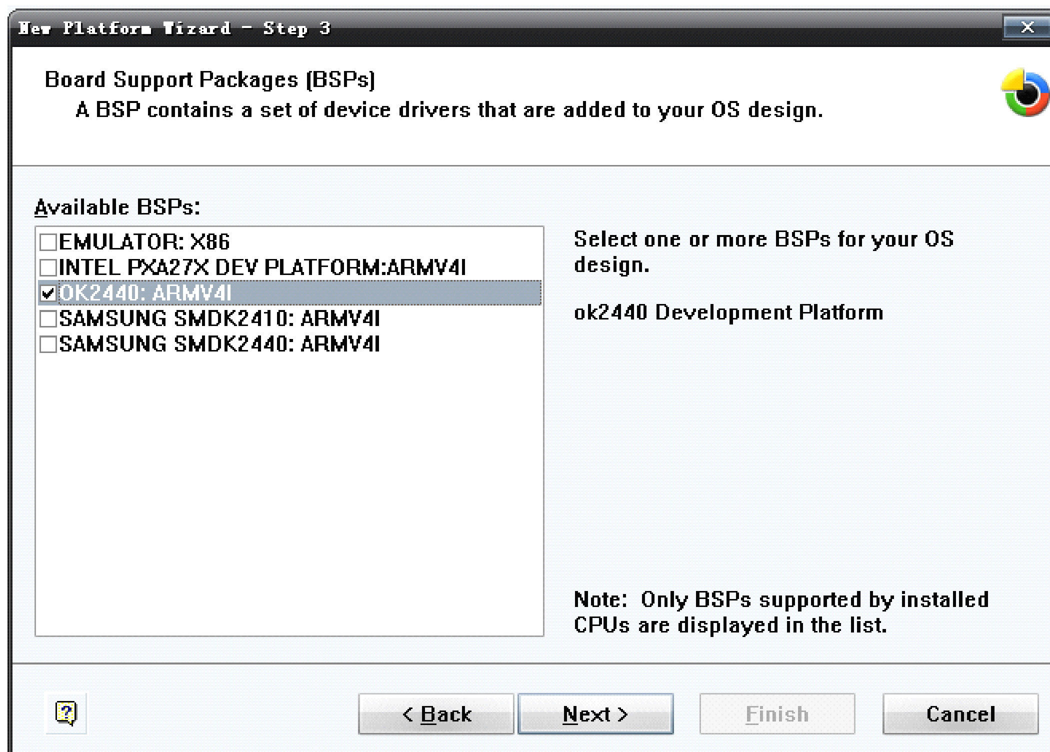


接着出现“New Platform Wizard - Step 2”框，在name框中输入平台的名称OK2440（可输入其他名字），点击“Next”按钮，如下图：

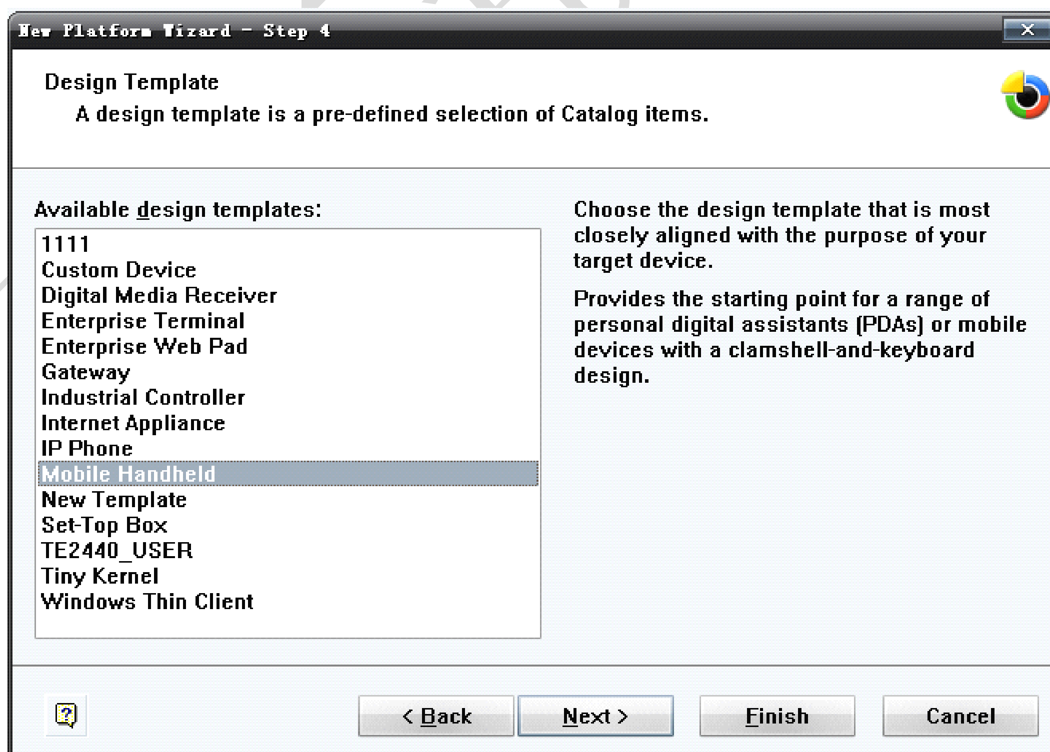


出现“New Platform Wizard - Step 3”框，您可看到“OK2440:ARMV4I” 的BSP。选中“OK2440:ARMV4I”

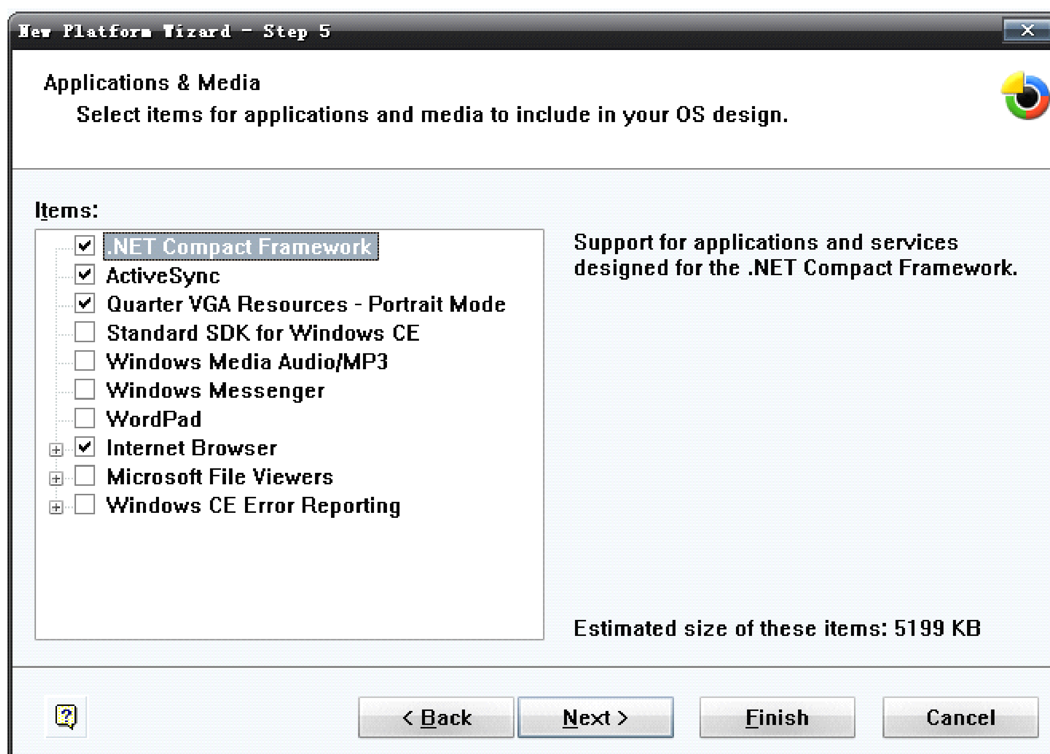
、再点击“Next” 按钮，如下图：



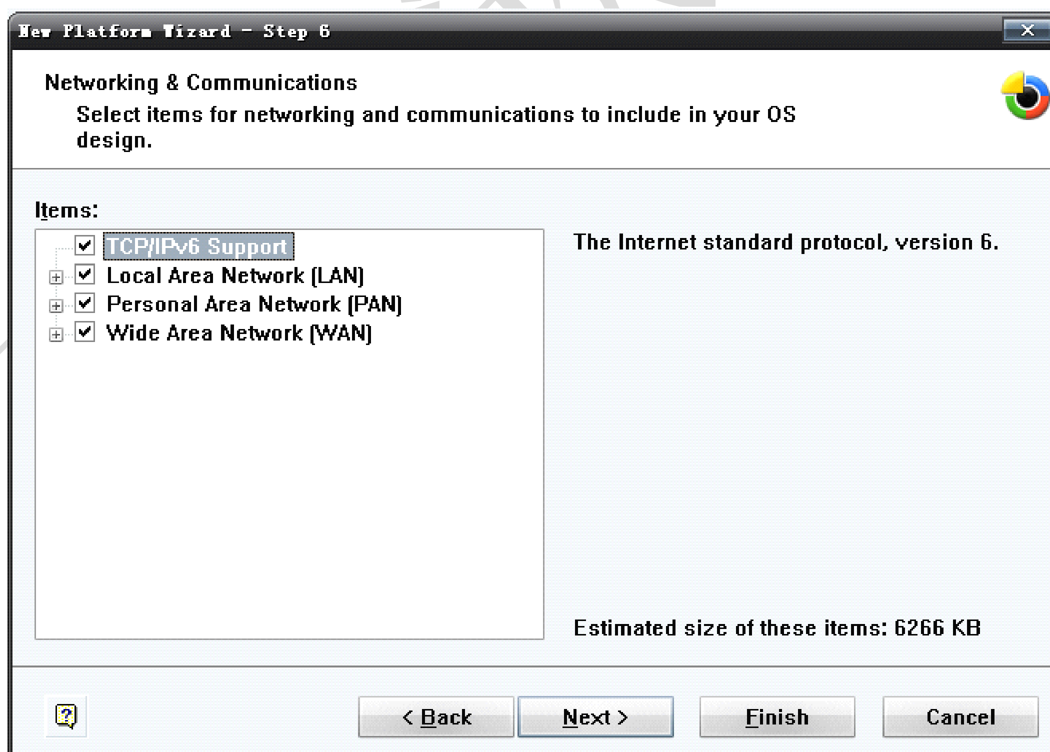
出现“New Platform Wizard - Step 4”对话框，在“Available design templates”列表中选择“Mobile Handheld”，点击“Next”按扭，如下 图：



接着出现“New Platform Wizard - Step 5”框，选择您需要的“Application & Media”，如下图:



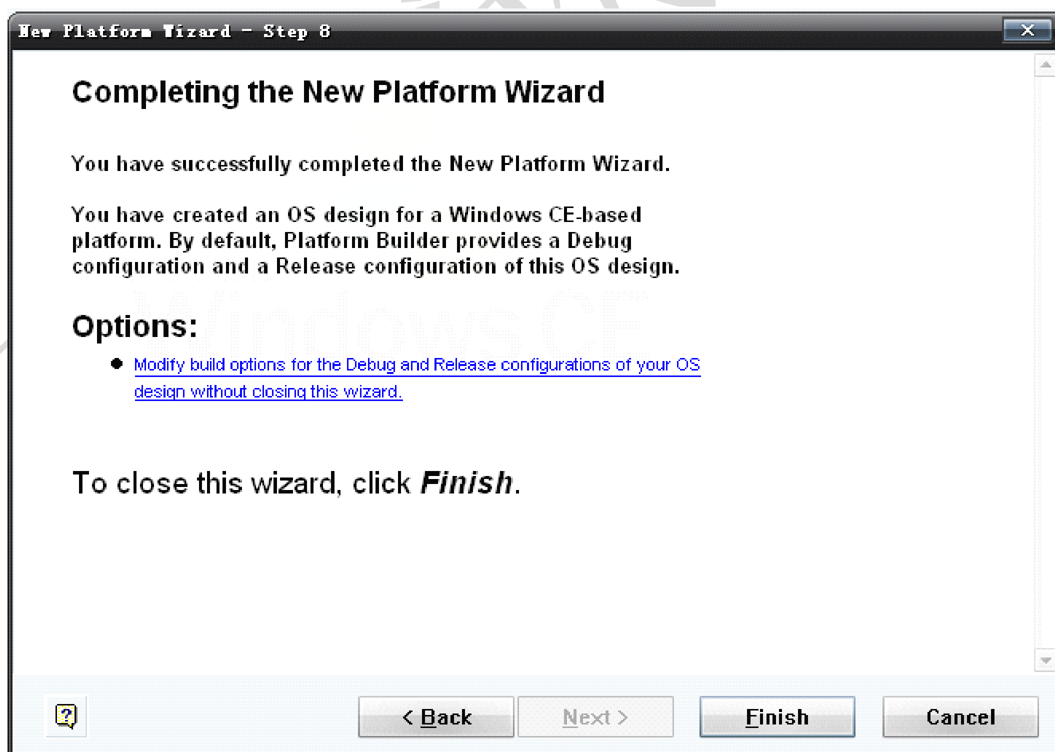
接着出现“New Platform Wizard - Step 6”框，选择您需要的“Networking & Communications”，如下图：



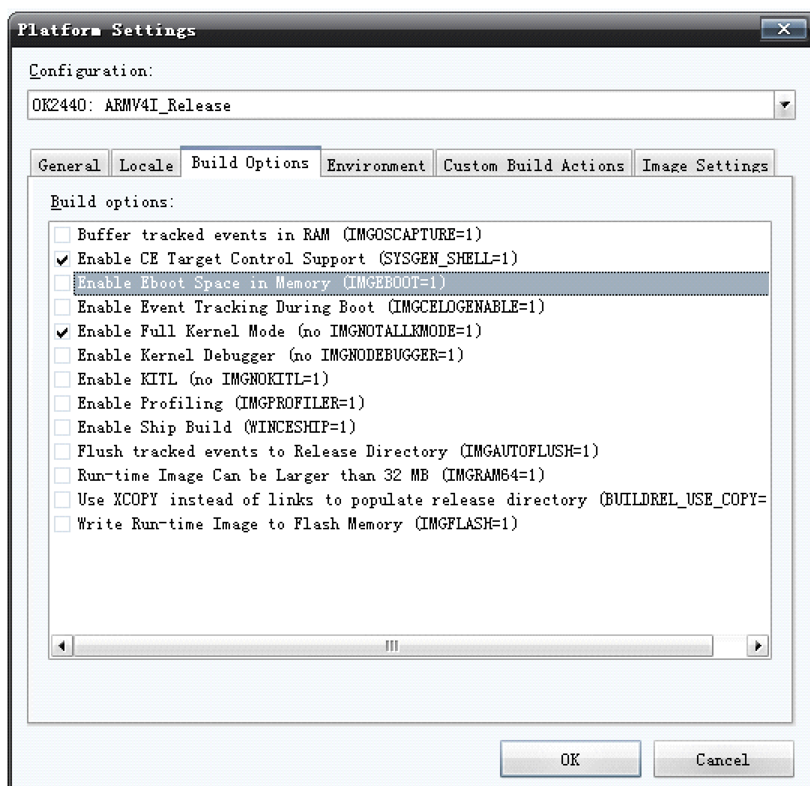
再点击”Next”按钮，出现以下的对话框。



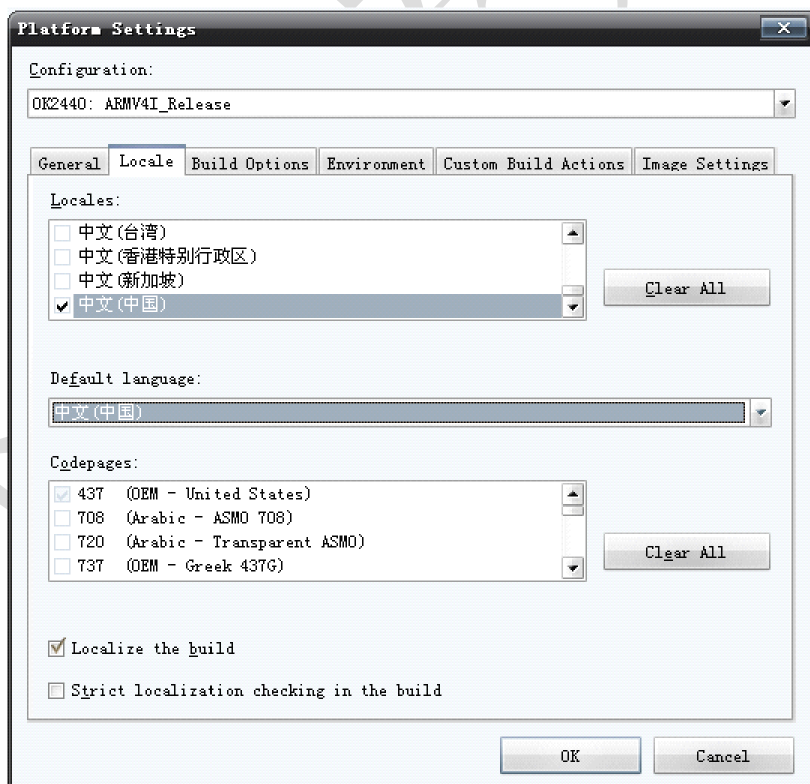
再点击“Next”按钮，您将看到“Completing the New Platform Wizard”对话框。建立新平台的所有设置步骤已经完成了。请点击“Finish”按钮。



下一步设置平台，点击 PB 的 Platform| Setting 菜单，在弹出的“Platform Settings”设置框的“Build Options”标签页，配置如下：



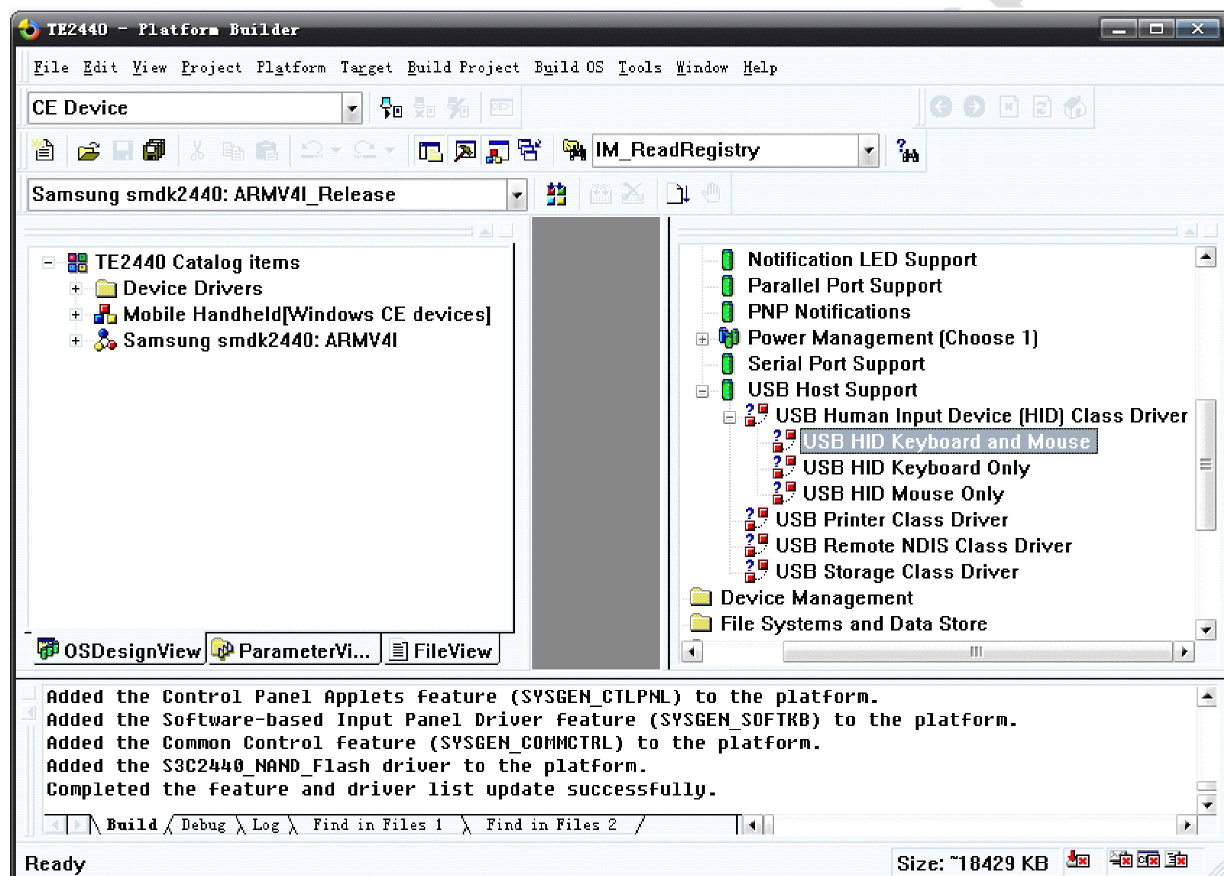
在“Locale”标签页设置中文支持：



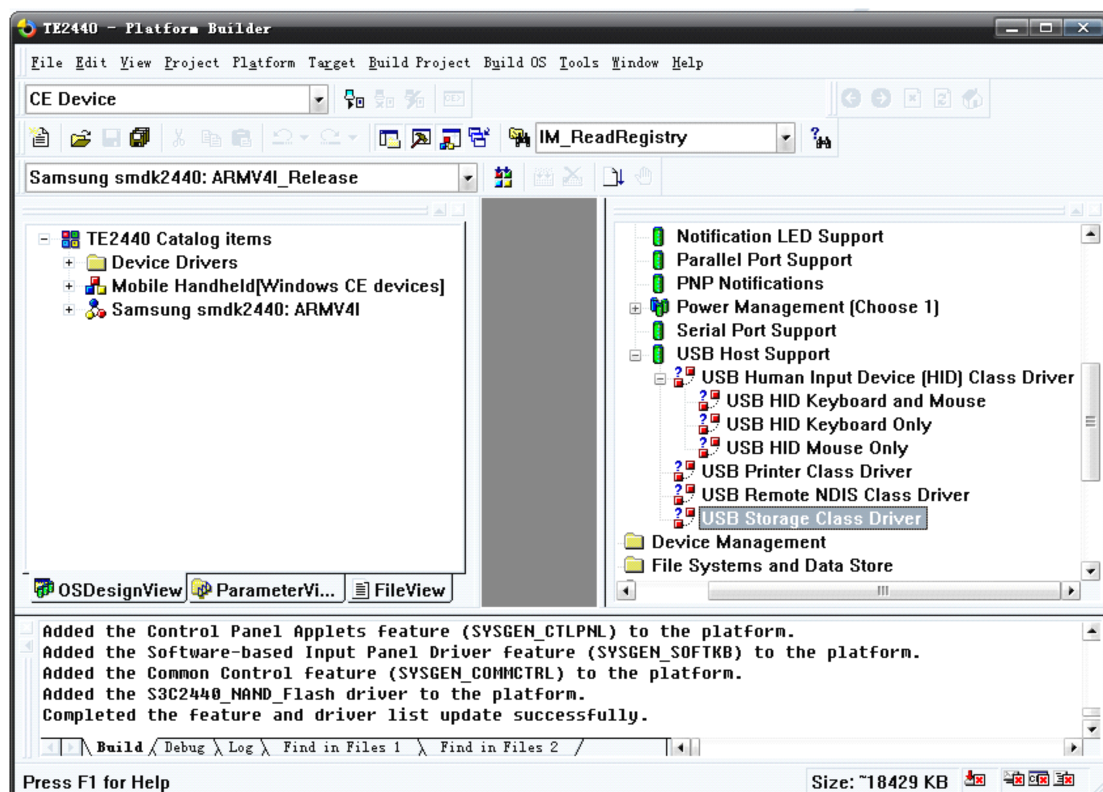
2.5 内核制定

要给新建的工程添加模块，展开右边的 Catalog 列表，在要添加的模块上点击右键，在右键菜单里选择 Add to OS Design 即可。

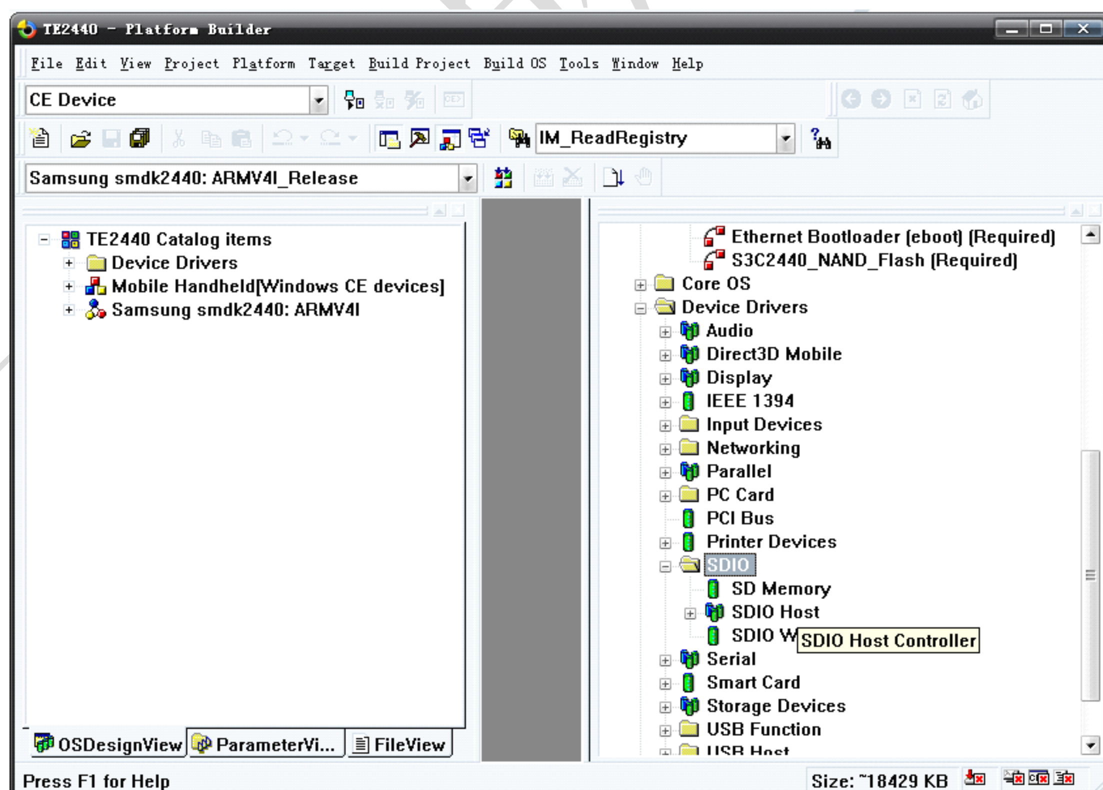
2.5.1 添加鼠标键盘支持



2.5.2 添加 U 盘支持

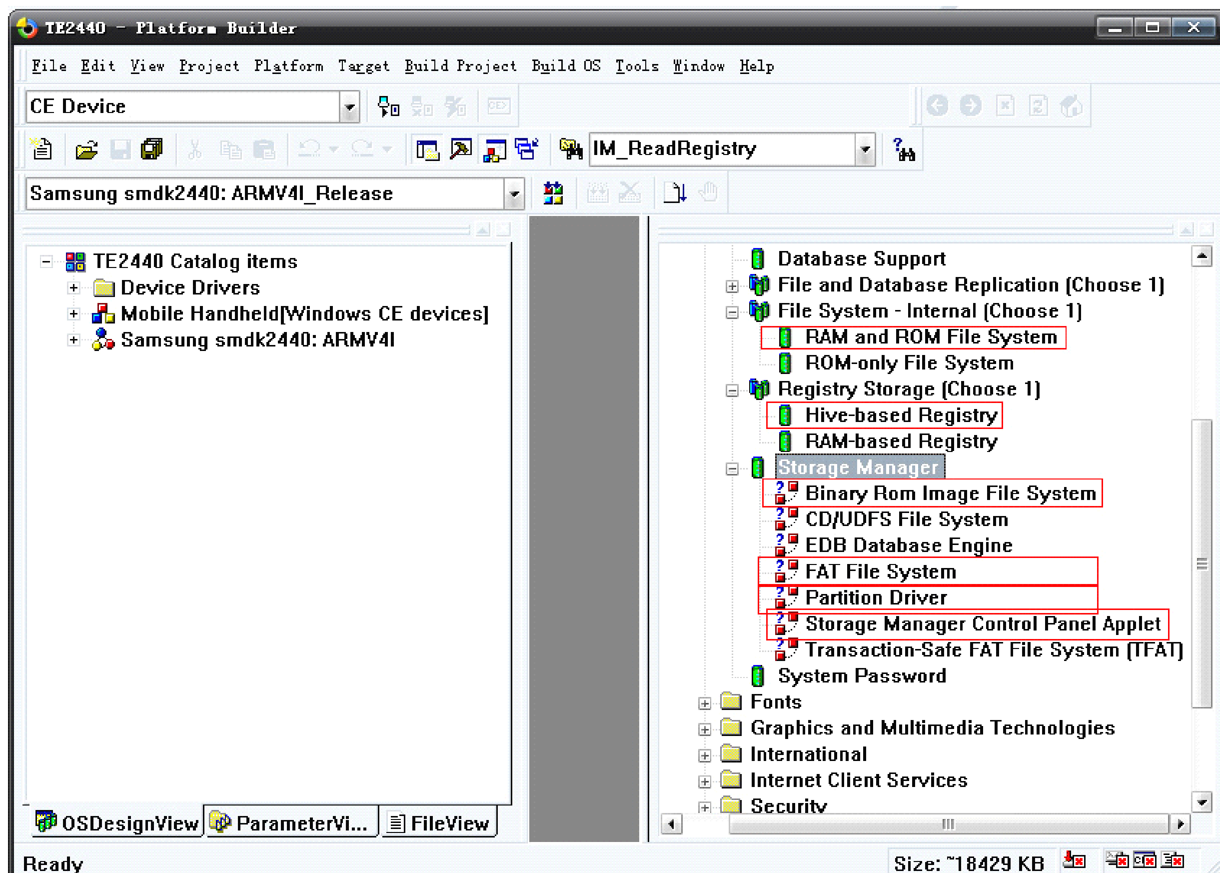

TE2440 - Platform Builder
File Edit View Project Platform Target Build Project Build OS Tools Window Help
CE Device
IM_ReadRegistry
Samsung smdk2440: ARMV4I_Release
TE2440 Catalog items
Device Drivers
Mobile Handheld[Windows CE devices]
Samsung smdk2440: ARMV4I
Notification LED Support
Parallel Port Support
PNP Notifications
Power Management (Choose 1)
Serial Port Support
USB Host Support
USB Human Input Device (HID) Class Driver
USB HID Keyboard and Mouse
USB HID Keyboard Only
USB HID Mouse Only
USB Printer Class Driver
USB Remote NDIS Class Driver
USB Storage Class Driver
Device Management
File Systems and Data Store
OSDesignView
ParameterVi...
FileView
Added the Control Panel Applets feature (SYSGEN_CTLPNL) to the platform.
Added the Software-based Input Panel Driver feature (SYSGEN_SOFTKB) to the platform.
Added the Common Control feature (SYSGEN_COMMCTRL) to the platform.
Added the S3C2440_NAND_Flash driver to the platform.
Completed the feature and driver list update successfully.
Build
Debug
Log
Find in Files 1
Find in Files 2
Press F1 for Help
Size: ~18429 KB

2.5.3 添加 SD 卡支持

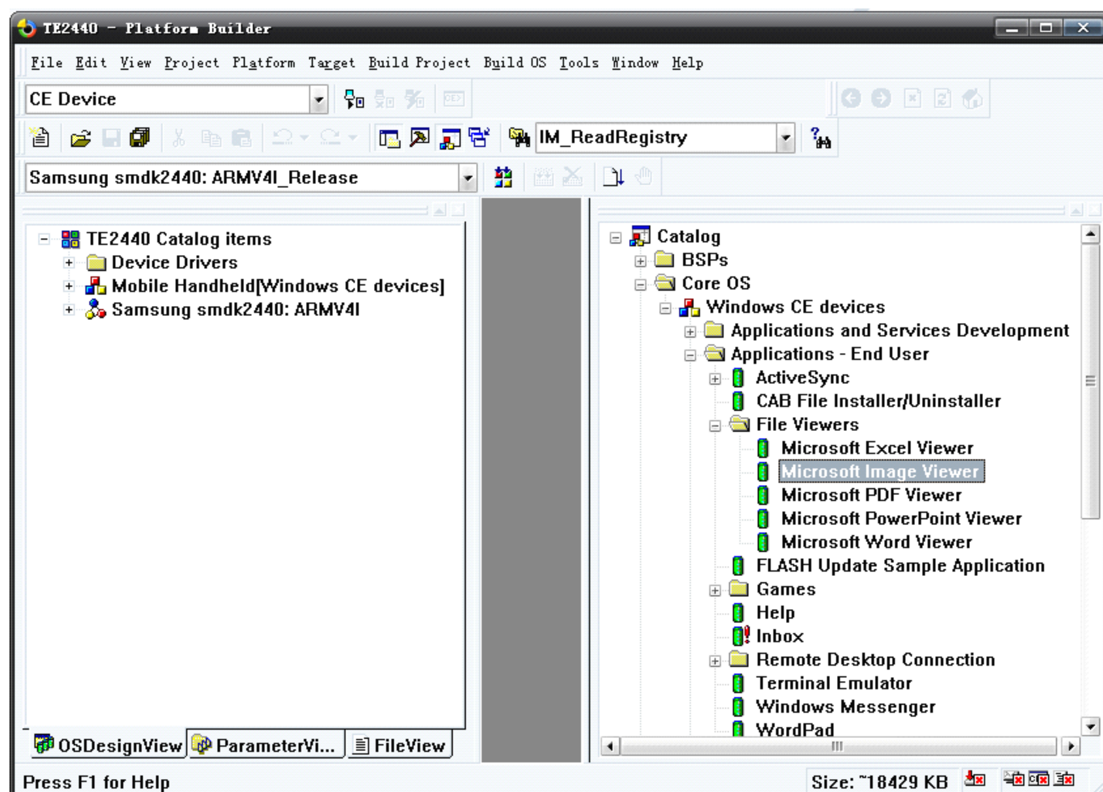

TE2440 - Platform Builder
File Edit View Project Platform Target Build Project Build OS Tools Window Help
CE Device
IM_ReadRegistry
Samsung smdk2440: ARMV4I_Release
TE2440 Catalog items
Device Drivers
Mobile Handheld[Windows CE devices]
Samsung smdk2440: ARMV4I
Ethernet Bootloader (eboot) (Required)
S3C2440_NAND_Flash (Required)
Core OS
Device Drivers
Audio
Direct3D Mobile
Display
IEEE 1394
Input Devices
Networking
Parallel
PC Card
PCI Bus
Printer Devices
SDIO
SD Memory
SDIO Host
SDIO Host Controller
Serial
Smart Card
Storage Devices
USB Function
OSDesignView
ParameterVi...
FileView
Press F1 for Help
Size: ~18429 KB

2.5.4 添加注册表保存功能

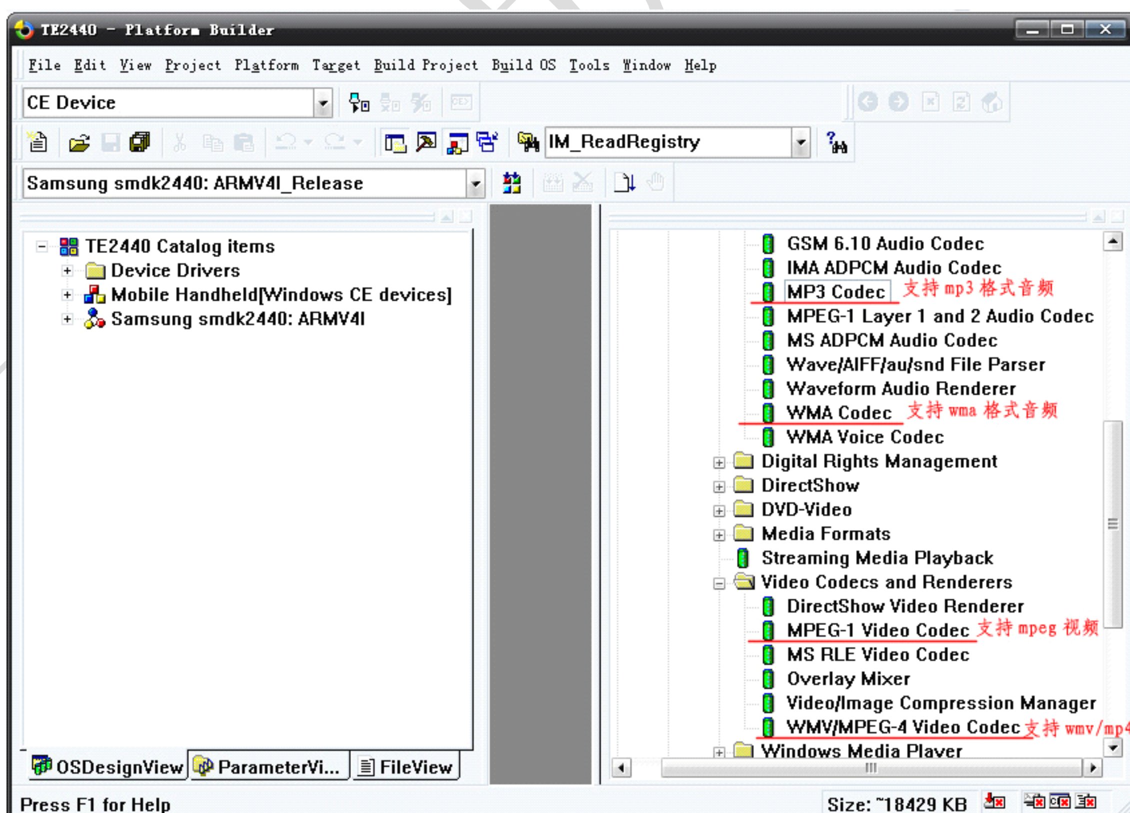
要实现保存注册表功能，需要添加如图所示模块：



2.5.5 添加图片浏览器

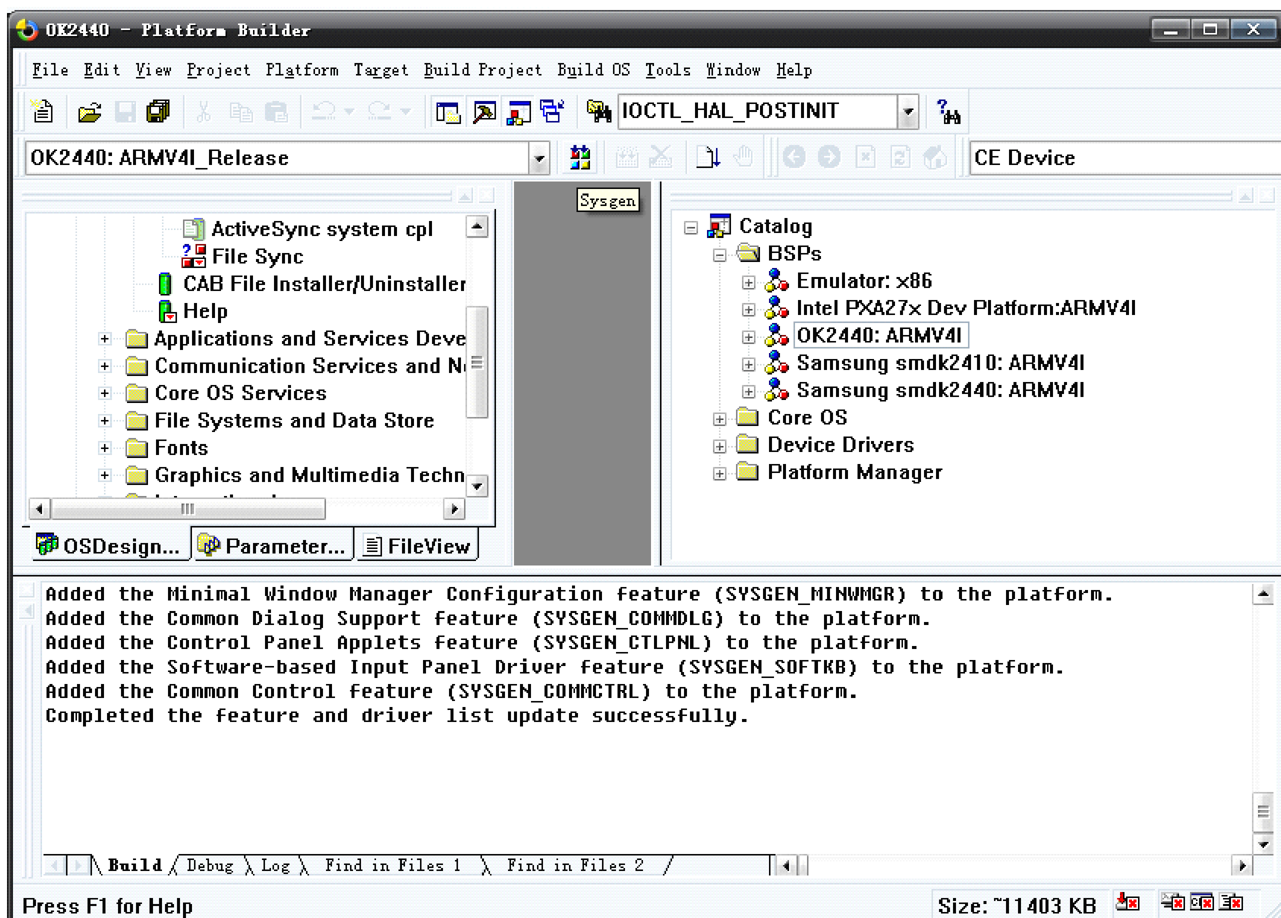


2.5.6 添加 MP3 MP4 支持



2.5.7 编译生成 Windows CE 映象文件

工程设置以及驱动添加完成后，点击 Build OS | Sysgen 开始编译平台，如图所示



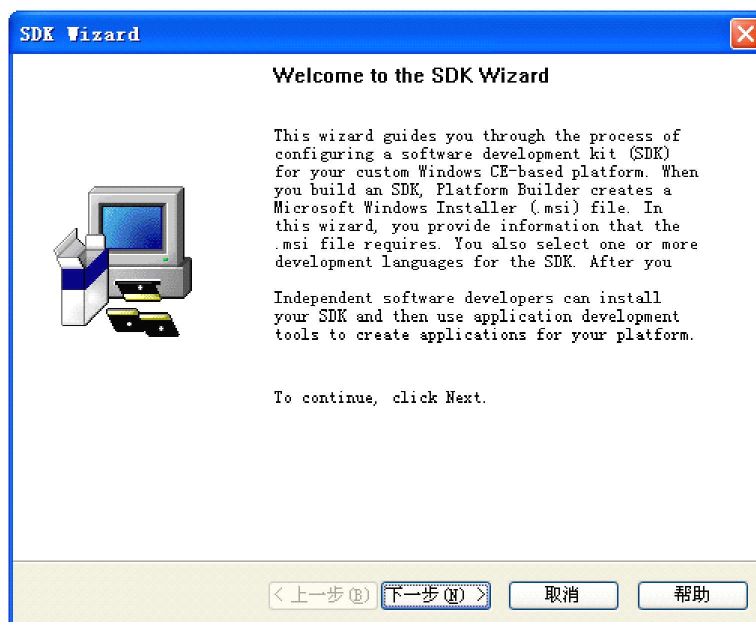
编译完成后，就生成了 OK2440 的二进制的 image：“nk.bin”和“nk.nb0”，一般而言，这两个文件位于编译平台时生成的文件夹：“WINCE500\PUBLIC\[PlatformName]\ RelDir\OK2440_ARMV4 Release”。

2.6 建立应用程序开发环境

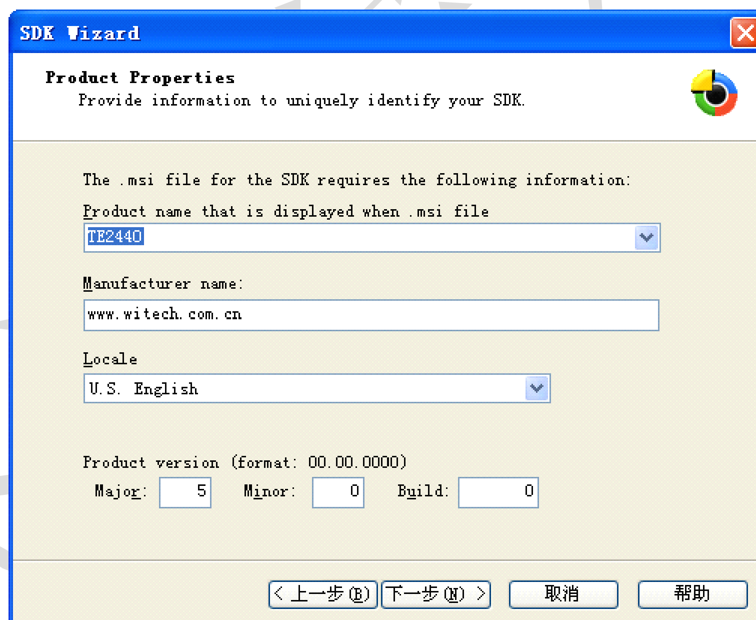
2.6.1 导出 SDK

启动 Platform Builder 5.0，打开已创建好的 OK2440 操作系统设计平台。

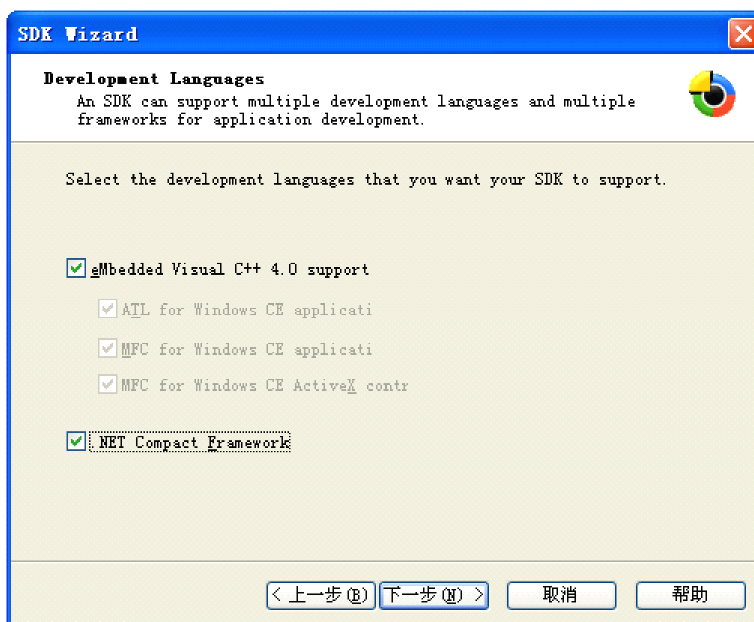
从“Platform”菜单上，选择“SDK”，然后选择“New SDK...”命令，出现如下图所示界面：



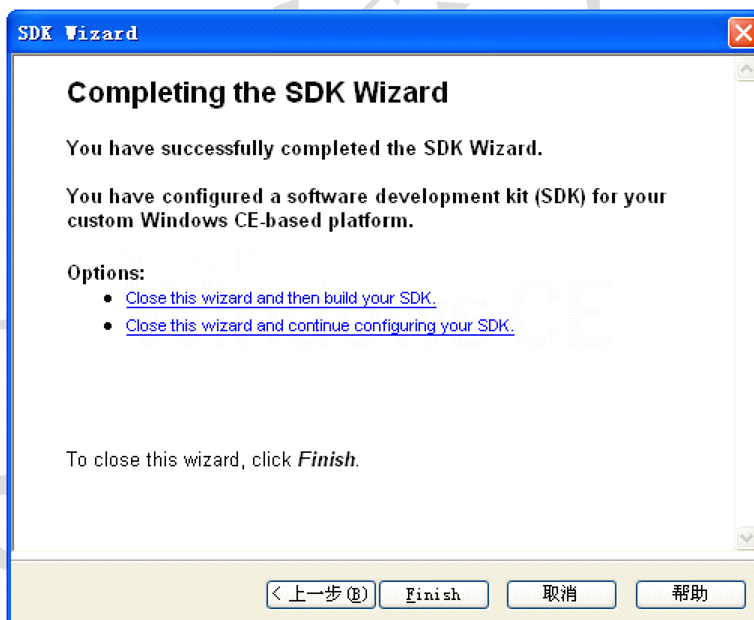
单击“下一步”按钮，在“Product Properties”对话框中，分别键入 SDK 名称和制造商名称：



单击“下一步”，在“Development Languages”对话框中选择对“eMbedded Visual C++ 4.0”和“.NET Compace Framework”的所有支持。



单击“下一步”，单击“Finish”按钮完成 SDK 向导。



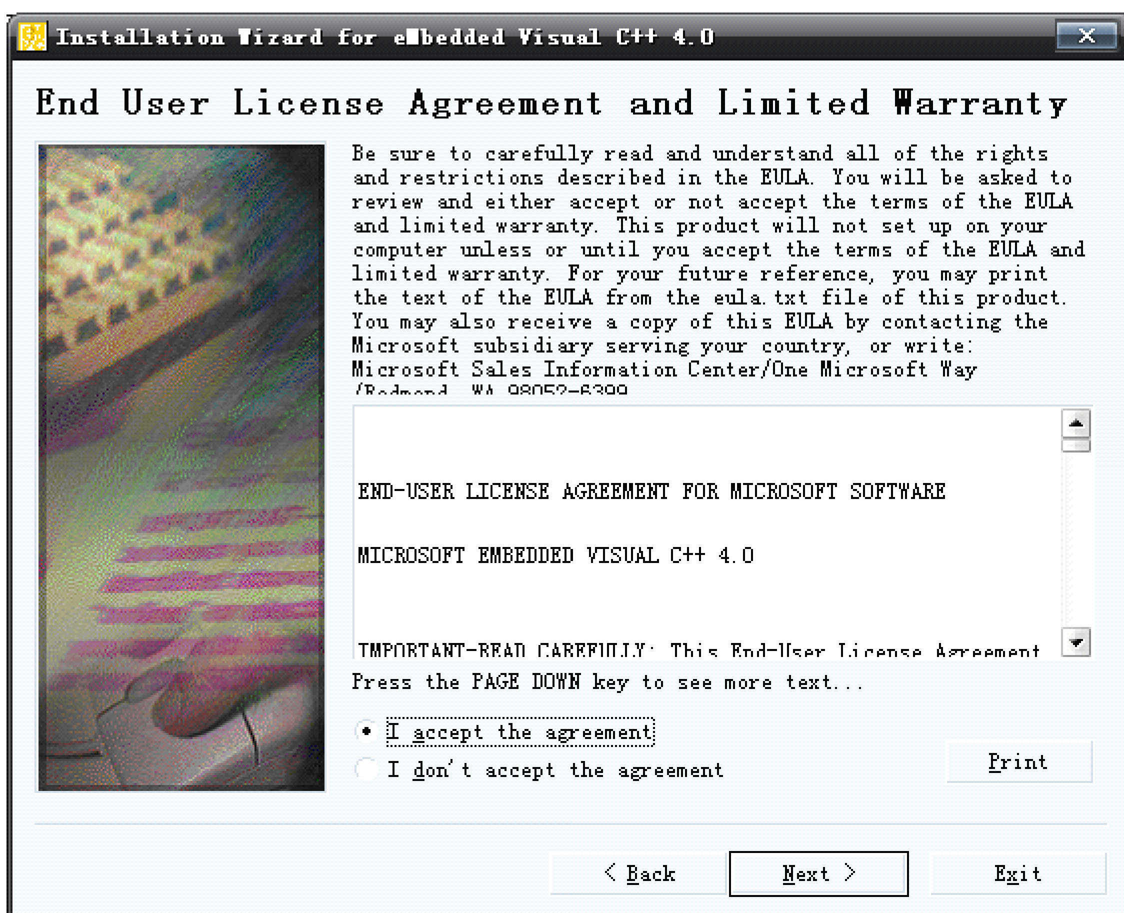
选择菜单“Platform-->SDK-->Build SDK...”，开始创建 SDK。成功后，创建的 SDK 为：E:\WINCE500\PBWorkspaces\OK2440\SDK 文件夹下的 OK2440_SDK.msi 文件。这是一个标准的 Windows installer 安装文件，双击鼠标即可安装。

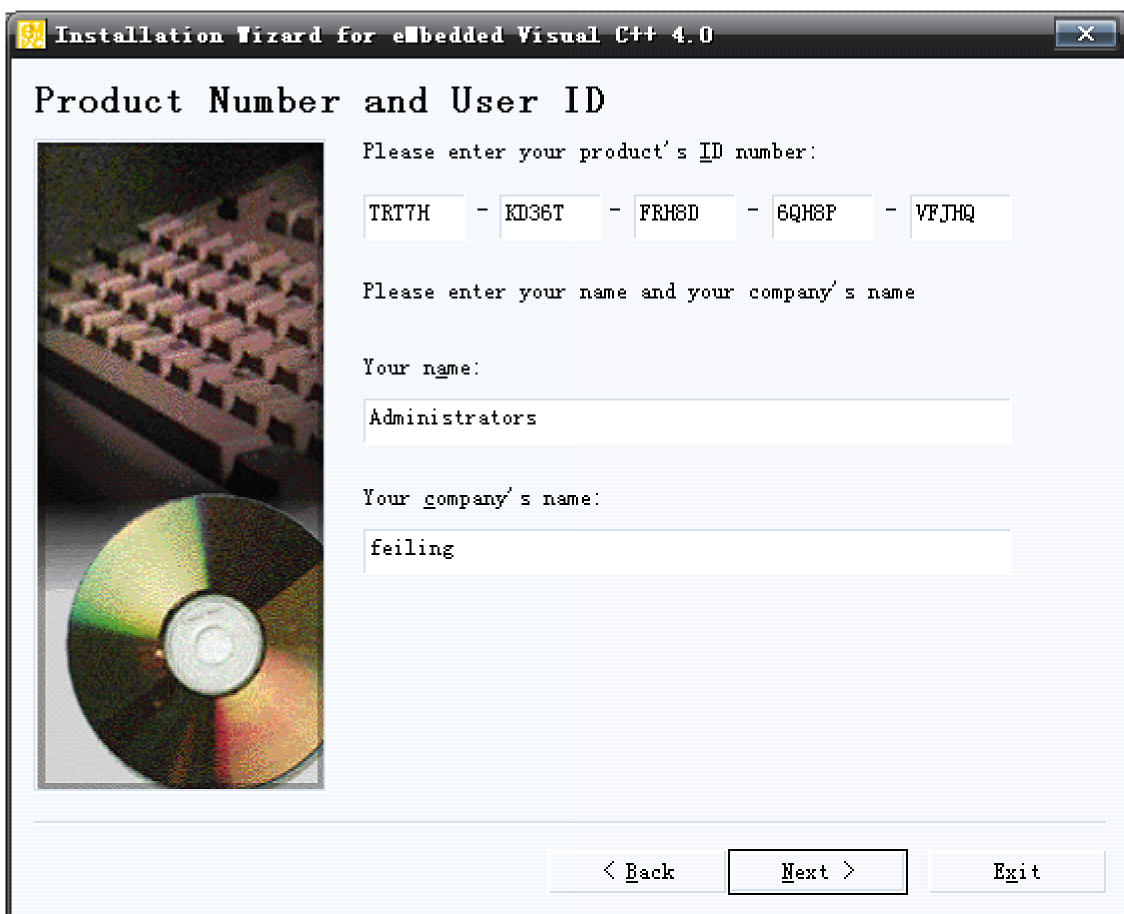
2.6.2 安装 eMbedded Visual C++ 4.0

eMbedded Visual C++简称 eVC，是用于创建 Windows CE 应用程序的一个集成开发环境，目前它的最新版本为 eMbedded Visual C++4.0 + Service Pack 4。（从 Microsoft Visual Studio .NET 2005 开始，微软不再更新 eMbedded Visual C++，所有 Windows CE 的应用程序开发都将在 Microsoft Visual Studio .NET 2005 中进行。）利用 eMbedded Visual C++，开发者可以为 Windows CE 开发 Win32，MFC，ATL 应用程序。

在用户光盘上找到 EVC 的安装程序，双击开始安装。




Installation Wizard for eMbedded Visual C++ 4.0
End User License Agreement and Limited Warranty
Be sure to carefully read and understand all of the rights and restrictions described in the EULA. You will be asked to review and either accept or not accept the terms of the EULA and limited warranty. This product will not set up on your computer unless or until you accept the terms of the EULA and limited warranty. For your future reference, you may print the text of the EULA from the eula.txt file of this product. You may also receive a copy of this EULA by contacting the Microsoft subsidiary serving your country, or write: Microsoft Sales Information Center/One Microsoft Way
END-USER LICENSE AGREEMENT FOR MICROSOFT SOFTWARE
MICROSOFT EMBEDDED VISUAL C++ 4.0
Press the PAGE DOWN key to see more text...
I accept the agreement
I don't accept the agreement
Print
< Back
Next >
Exit

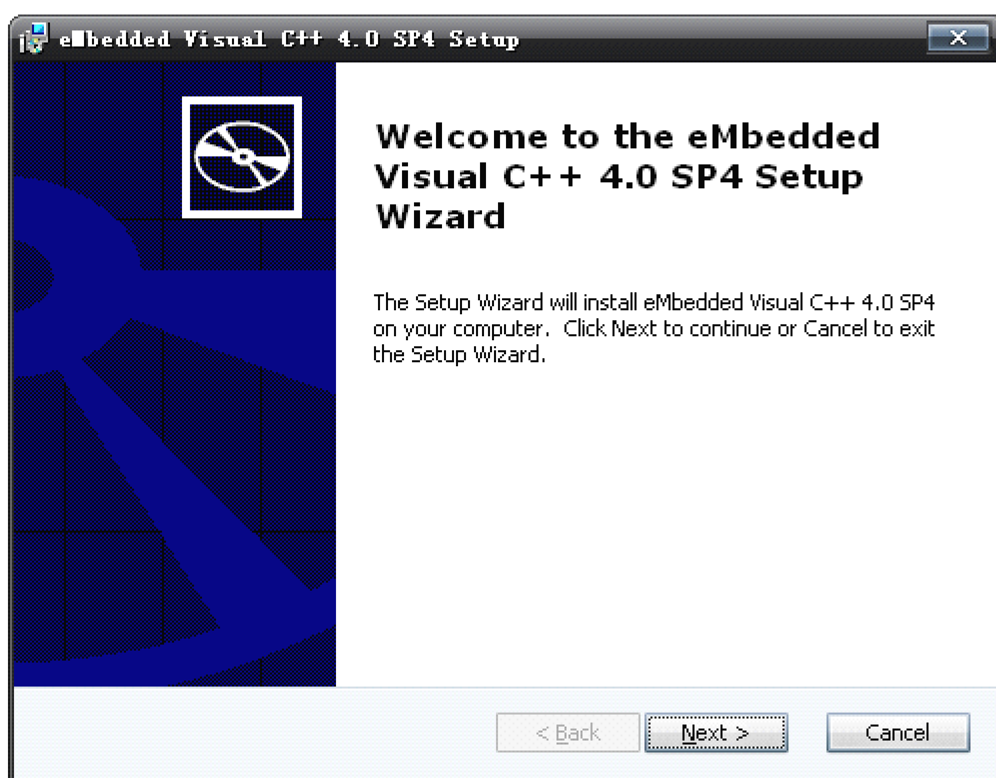


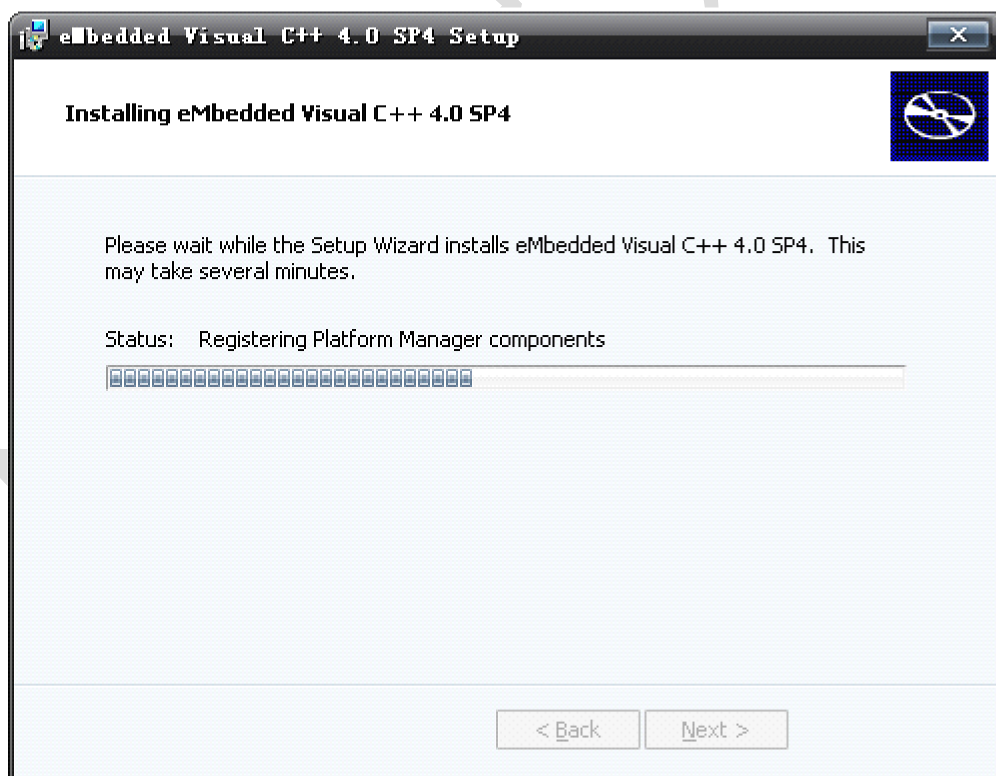
2.6.3 安装 SP4

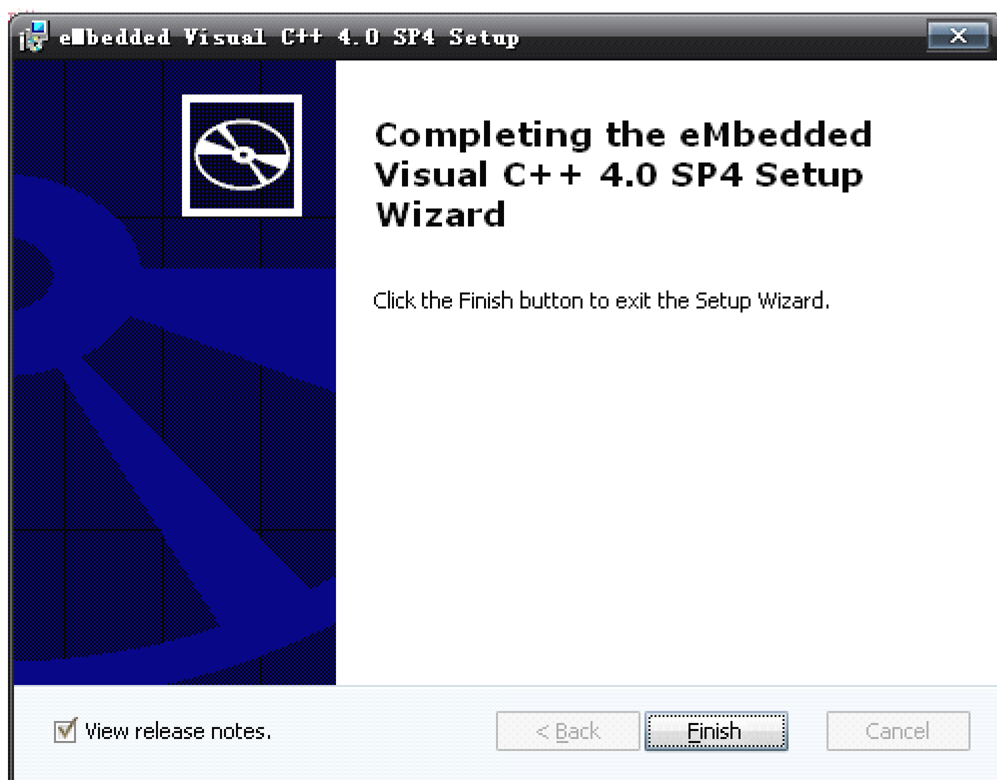
安装完 EVC4.0 后，还需要安装 Service Pack 4。下面是 eVC 的版本与可开发的 Windows CE 应用程序版本的对应。

eVC 版本	可开发的 Windows CE 应用程序版本
eMbedded Visual C++ 3.0	Windows CE 3.0
eMbedded Visual C++ 4.0	Windows CE .NET 4.0
eMbedded Visual C++ 4.0 + SP1	Windows CE .NET 4.1
eMbedded Visual C++ 4.0 + SP2/SP3	Windows CE .NET 4.2
eMbedded Visual C++ 4.0 + SP4	Windows CE 5.0

找到 SP4 的安装程序 OK2440\wince\evc4sp4.exe，双击开始安装。


eMbedded Visual C++ 4.0 SP4 Setup
Welcome to the eMbedded Visual C++ 4.0 SP4 Setup Wizard
The Setup Wizard will install eMbedded Visual C++ 4.0 SP4 on your computer. Click Next to continue or Cancel to exit the Setup Wizard.
< Back
Next >
Cancel

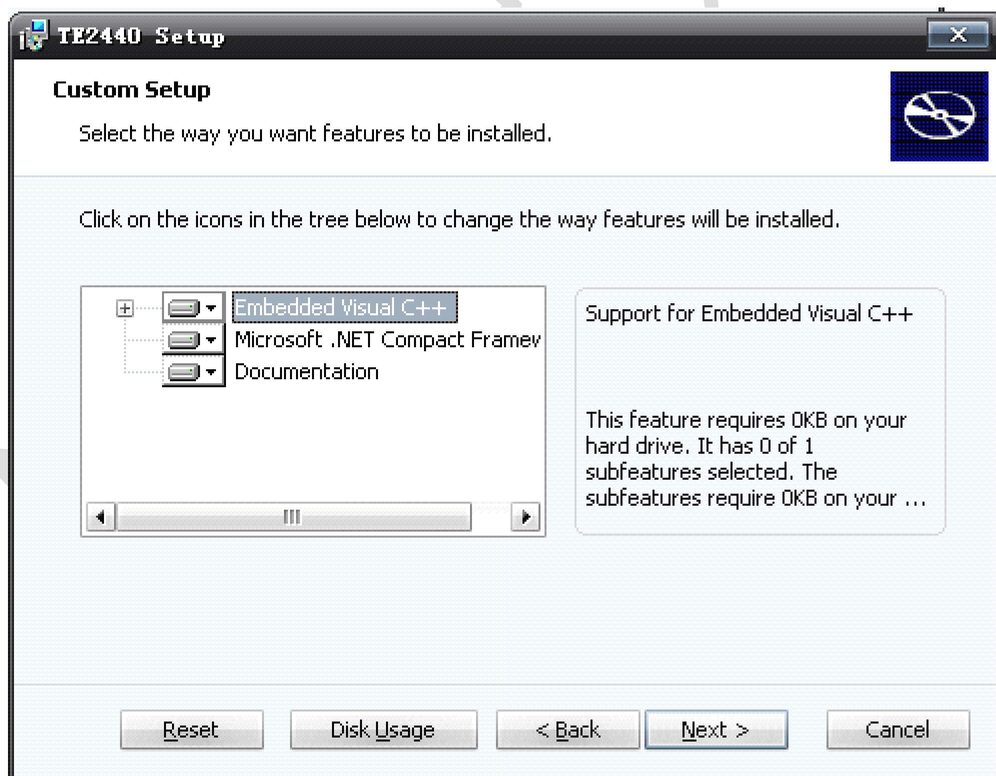

eMbedded Visual C++ 4.0 SP4 Setup
Installing eMbedded Visual C++ 4.0 SP4
Please wait while the Setup Wizard installs eMbedded Visual C++ 4.0 SP4. This may take several minutes.
Status: Registering Platform Manager components
< Back
Next >



2.6.4 安装 SDK

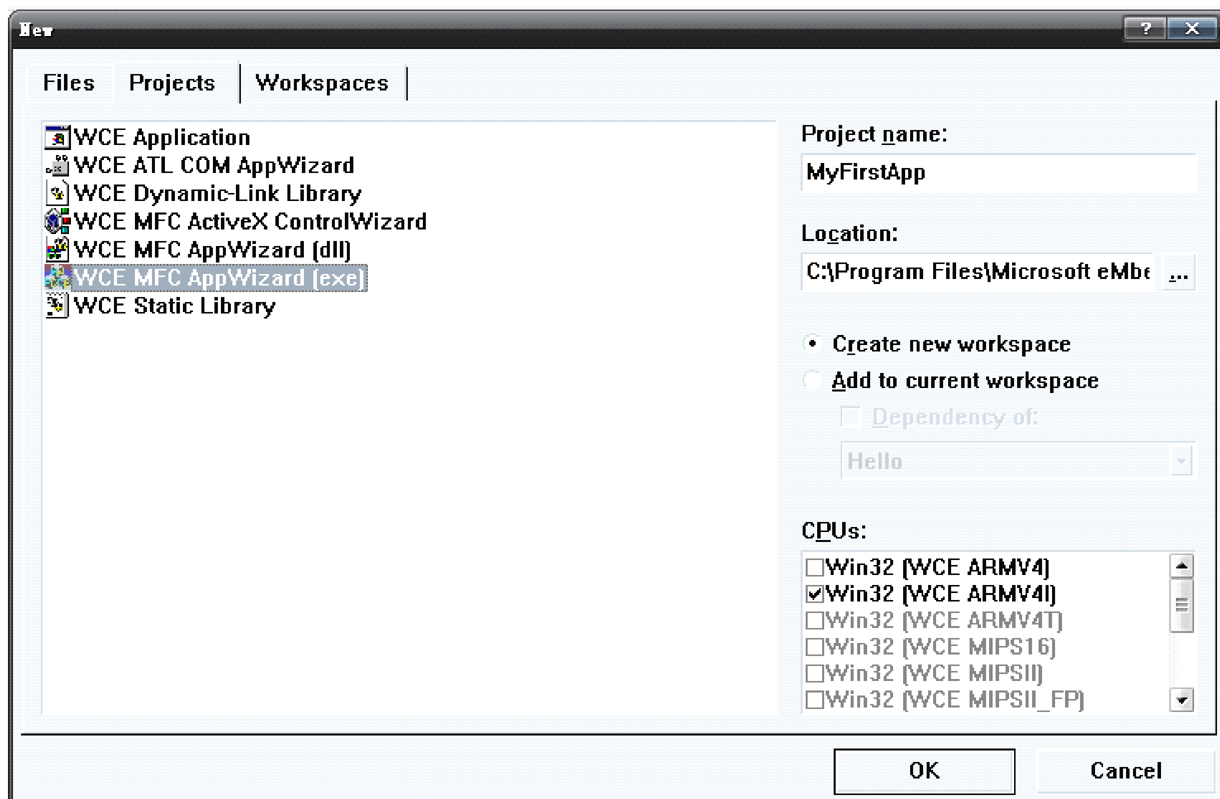
以上工作完成后，接下来安装 7.6.1 节导出的 SDK。(用户光盘中也提供了一个 SDK 安装程序，若不想自己导出的话可以直接用那个)

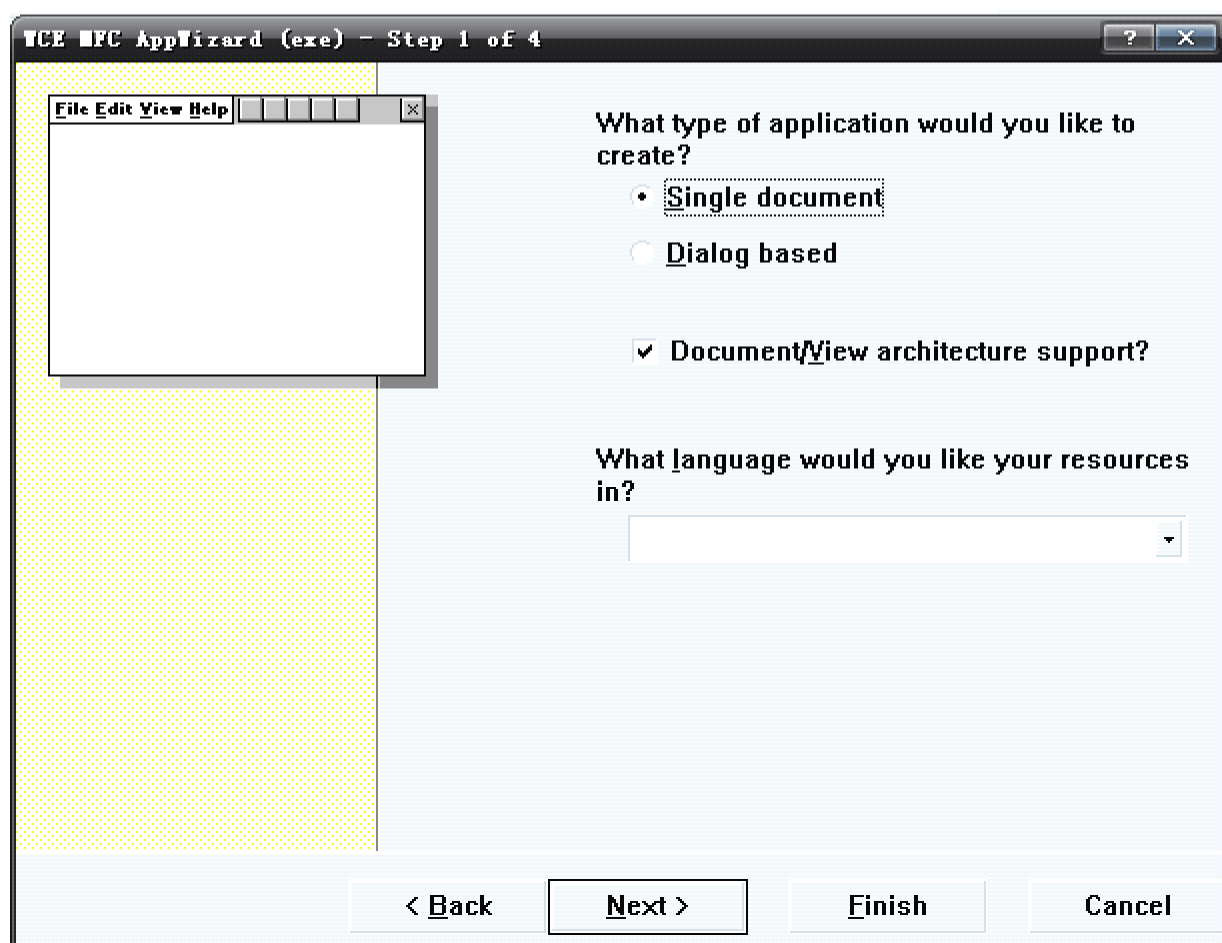

TE2440 Setup
Welcome to the TE2440 Setup Wizard
The Setup Wizard will allow you to change the way TE2440 features are installed on your computer or even to remove TE2440 from your computer. Click Next to continue or Cancel to exit the Setup Wizard.
< Back
Next >
Cancel

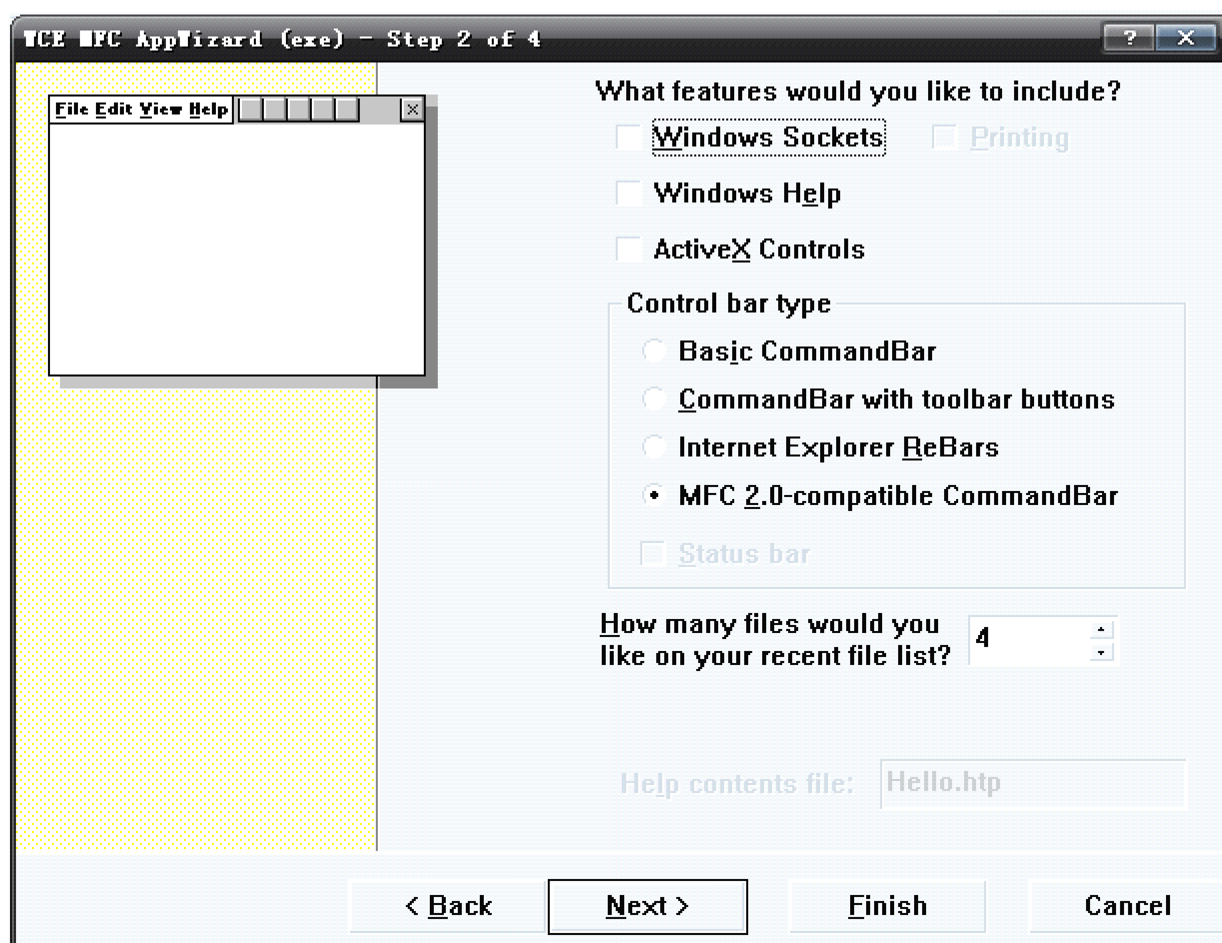

TE2440 Setup
Custom Setup
Select the way you want features to be installed.
Click on the icons in the tree below to change the way features will be installed.
Embedded Visual C++
Microsoft .NET Compact Framev
Documentation
Support for Embedded Visual C++
This feature requires 0KB on your hard drive. It has 0 of 1 subfeatures selected. The subfeatures require 0KB on your ...
Reset
Disk Usage
< Back
Next >
Cancel

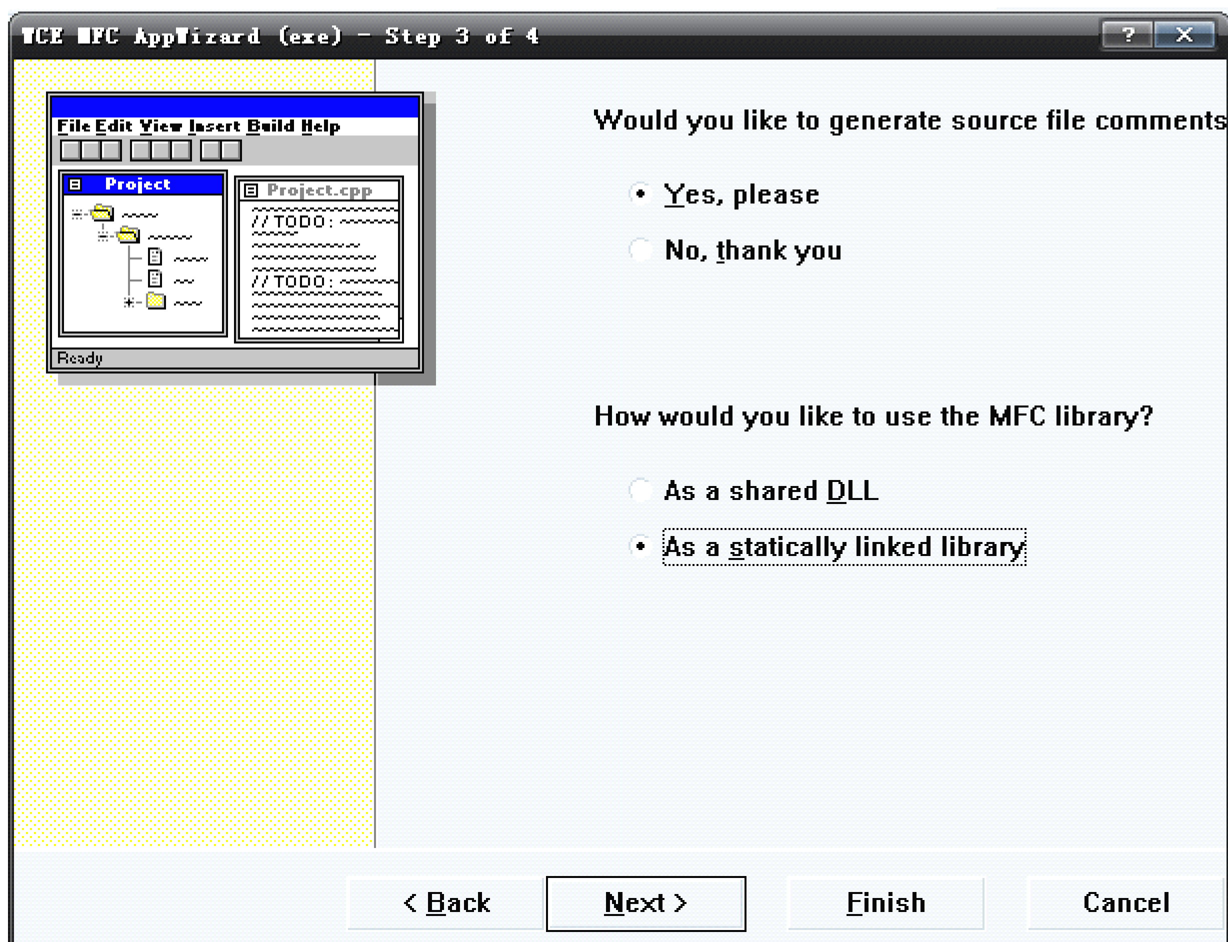
2.7 Wince 应用程序开发示例

建立好应用程序开发环境之后，打开 EVC4.0，点击 file 菜单的 new 按钮，按照下图设置，这里我们新建一个基于 MFC 的应用程序。



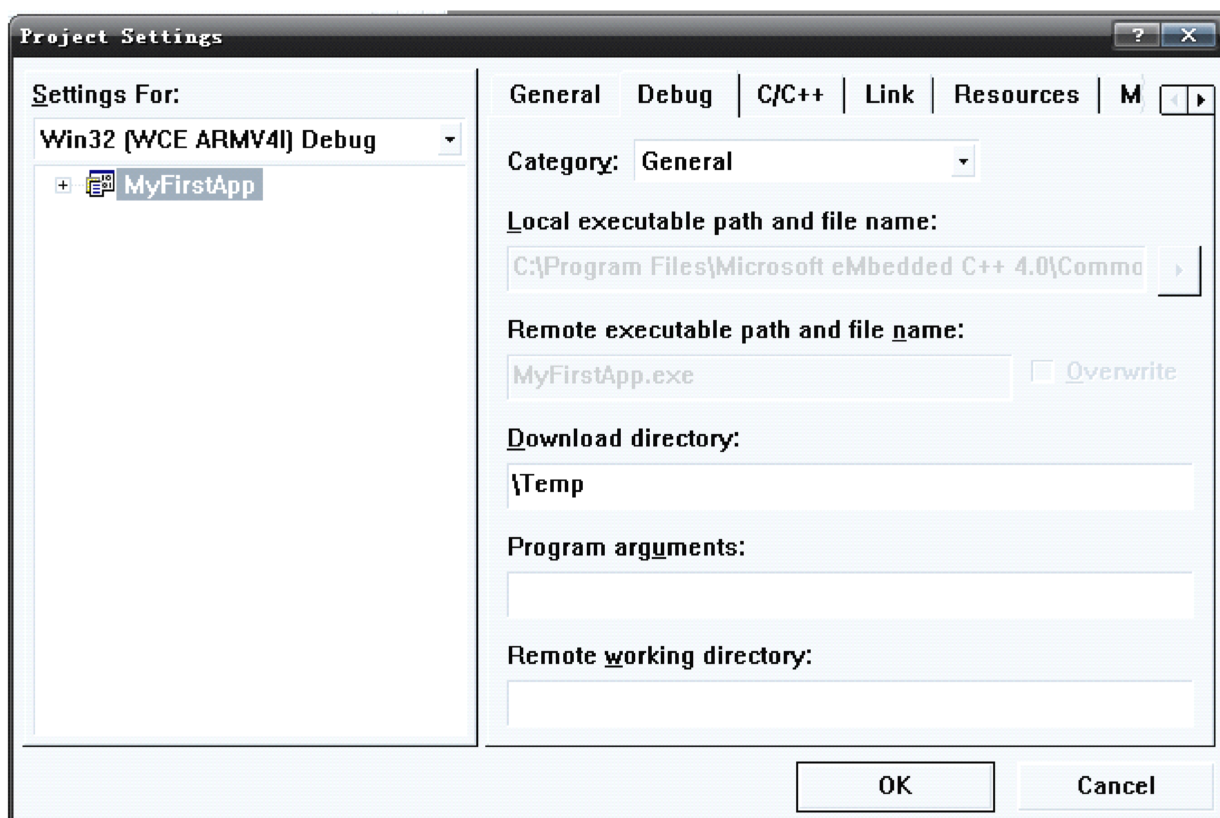

WCE MFC AppWizard (exe) - Step 1 of 4
File Edit View Help
What type of application would you like to create?
Single document
Dialog based
Document/View architecture support?
What language would you like your resources in?
< Back
Next >
Finish
Cancel


WCE MFC AppWizard (exe) - Step 2 of 4
File Edit View Help
What features would you like to include?
Windows Sockets
Printing
Windows Help
ActiveX Controls
Control bar type
Basic CommandBar
CommandBar with toolbar buttons
Internet Explorer ReBars
MFC 2.0-compatible CommandBar
Status bar
How many files would you like on your recent file list?
4
Help contents file:
Hello.htp
< Back
Next >
Finish
Cancel



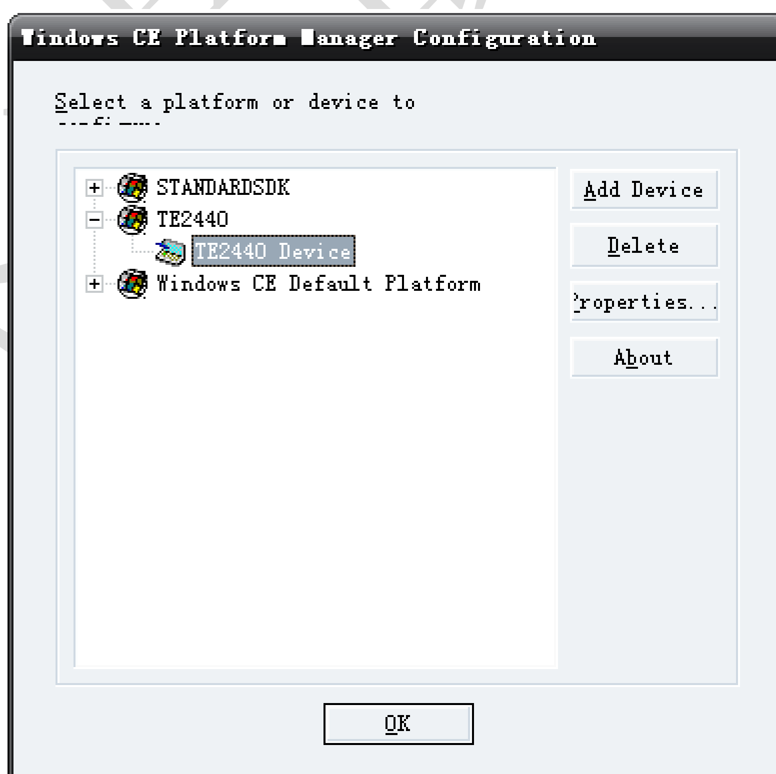
建好后，点击 Project 菜单的 Settings 按钮可对工程进行设置。

Debug 面板上的 Download directory 为部署到 wince（开发板）的目录。这里我们将其改为\Temp，点击 OK 完成设置。

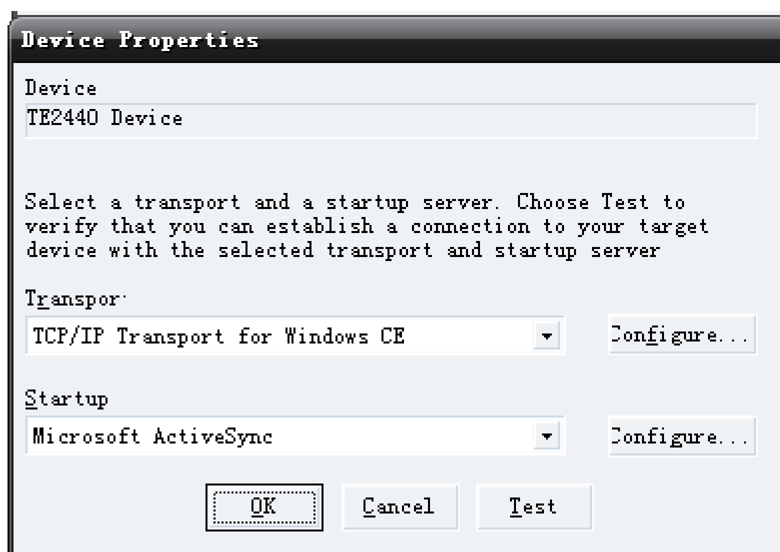


点击“tools”菜单的“Configure platform manager”菜单项进行平台管理器的配置：

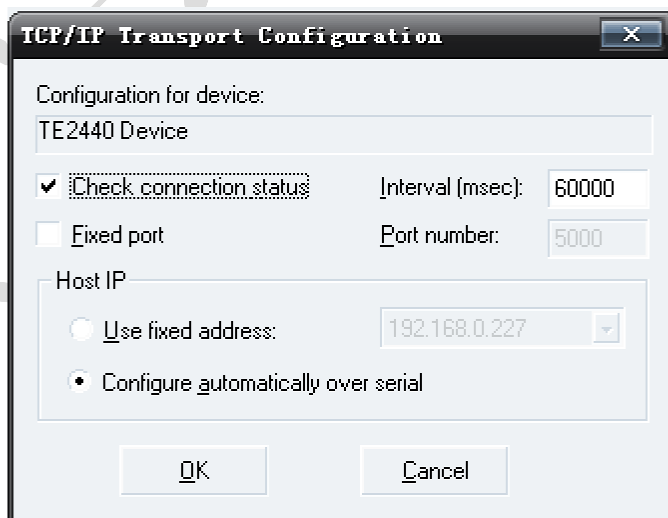
在弹出的对话框中添加一个新设备，设备名可以取为“OK2440”，如下图



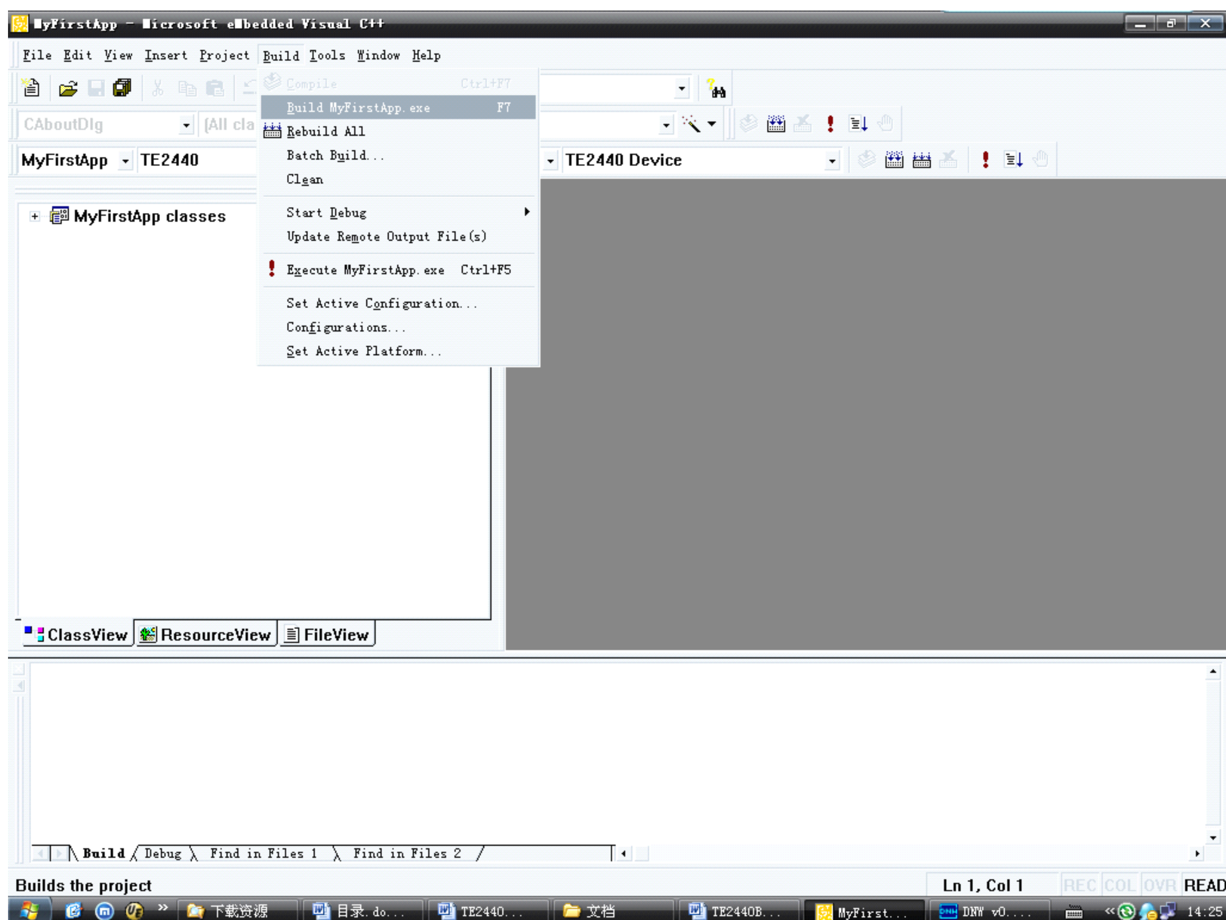
点击右边的“Propertions…”,设置“OK2440”设备的属性，如下图



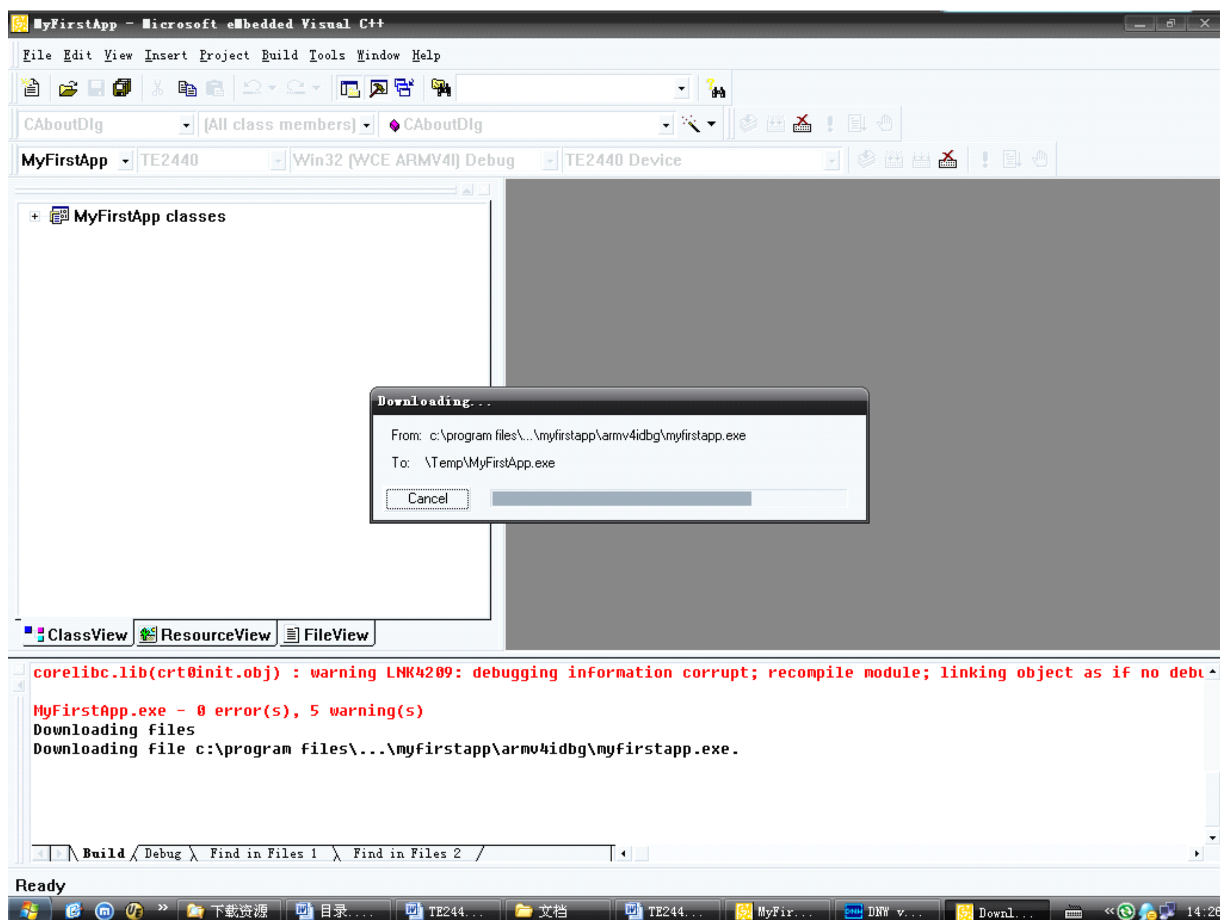
点击“Transport”下拉框右边的“Configure”按钮，以设置 TCP/TP 传输的设置，按下图所示进行配置。



以上配置都完成后按 F7 进行编译。



编译成功后会自动部署到开发板上 wince 的\Temp 文件夹。


MyFirstApp - Microsoft eMbedded Visual C++
File Edit View Insert Project Build Tools Window Help
CAboutDlg
[All class members]
CAboutDlg
MyFirstApp
TE2440
Win32 (WCE ARMV4I) Debug
TE2440 Device
MyFirstApp classes
Downloading...
From: c:\program files\...\myfirstapp\armv4idbg\myfirstapp.exe
To: \Temp\MyFirstApp.exe
Cancel
ClassView
ResourceView
FileView
corelibc.lib(crt0init.obj) : warning LNK4209: debugging information corrupt; recompile module; linking object as if no debu
MyFirstApp.exe - 0 error(s), 5 warning(s)
Downloading files
Downloading file c:\program files\...\myfirstapp\armv4idbg\myfirstapp.exe.
Build
Debug
Find in Files 1
Find in Files 2
Ready
14:26

附录一 CE 下访问物理地址的方法

嵌入式设备与桌面 PC 的一个显著不同是它的应用程序中通常需要直接访问某一段物理内存，这在驱动程序中对物理内存的访问尤为重要，尤其是像 ARM 体系结构下，I/O 端口也被映射成某一个物理内存地址。因此，与桌面版本 Windows 相比，Windows CE 提供了相对简单的物理内存访问方式。无论是驱动程序还是应用程序都可以通过 API 访问某一段物理内存。

Windows CE 的有些函数中需要用到物理内存结构体 PHYSICAL_ADDRESS， Windows CE 在 ceddk.h 中定义了 PHYSICAL_ADDRESS，它其实是 LARGE_INTEGER 类型，其定义如下：

```
// in ceddk.h
typedef LARGE_INTEGER PHYSICAL_ADDRESS, *PPHYSICAL_ADDRESS;
// in winnt.h
typedef union _LARGE_INTEGER{
struct{
DWORD LowPart;
LONG HighPart;
};
LONGLONG QuadPart;
} LARGE_INTEGER;
```

可见，Windows CE 中用 64 个 Bit 来代表物理地址，对于大多数 32 位的 CPU 而言，只需要把它的 HighPart 设置为 0 就可以了。

如果要直接访问某一个地址的物理内存， Windows CE 提供了 VirtualAlloc()和 VirtualCopy() 函数，VirtualAlloc 负责在虚拟内存空间内保留一段虚拟内存，而 VirtualCopy 负责把一段物理内存和虚拟内存绑定，这样，最终对物理内存的访问还是通过虚拟地址进行。它们的声明如下：

```
// 申请虚拟内存
LPVOID VirtualAlloc(
LPVOID lpAddress, // 希望的虚拟内存起始地址
DWORD dwSize, // 以字节为单位的大小
DWORD flAllocationType, // 申请类型，分为 Reserve 和 Commit
DWORD flProtect // 访问权限
```

```
);
// 把物理内存绑定到虚拟地址空间
BOOL VirtualCopy(
LPVOID lpvDest, // 虚拟内存的目标地址
LPVOID lpvSrc, // 物理内存地址
DWORD cbSize, // 要绑定的大小
DWORD fdwProtect // 访问权限
);
```

VirtualAlloc 对虚拟内存的申请分为两步，保留 MEM_RESERVE 和提交 MEM_COMMIT。其中 MEM_RESERVE 只是在进程的虚拟地址空间内保留一段，并不分配实际的物理内存，因此保留的虚拟内存并不能被应用程序直接使用。MEM_COMMIT 阶段才真正的为虚拟内存分配物理内存。

下面的代码显示了如何使用 VirtualAlloc 和 VirtualCopy 来访问物理内存。因为 VirtualCopy 负责把一段物理内存和虚拟内存绑定，所以 VirtualAlloc 的时候只需要对内存保留，没有必要提交。

```
FpDriverGlobals =
(PDRIVER_GLOBALS) VirtualAlloc(
0,
DRIVER_GLOBALS_PHYSICAL_MEMORY_SIZE,
MEM_RESERVE,
PAGE_NOACCESS);
if (FpDriverGlobals == NULL) {
ERRORMSG(DRIVER_ERROR_MSG, (TEXT(" VirtualAlloc failed!\r\n")));
return;
}
else {
if (!VirtualCopy(
(PVOID)FpDriverGlobals,
(PVOID)(DRIVER_GLOBALS_PHYSICAL_MEMORY_START),
DRIVER_GLOBALS_PHYSICAL_MEMORY_SIZE,
(PAGE_READWRITE | PAGE_NOCACHE))) {
```

```
ERRORMSG(DRIVER_ERROR_MSG, (TEXT("VirtualCopy failed!\r\n")));
return;
}
}
```

CEDDK 还提供了函数 MmMapIoSpace 用来把一段物理内存直接映射到虚拟内存。此函数的原形如下：

```
PVOID MmMapIoSpace(
PHYSICAL_ADDRESS PhysicalAddress, // 起始物理地址
ULONG NumberOfBytes, // 要映射的字节数
BOOLEAN CacheEnable // 是否缓存
);
```

其实，MmMapIoSpace 函数内部也是调用 VirtualAlloc 和 VirtualCopy 函数来实现物理地址到虚拟地址的映射的。MmMapIoSpace 函数的原代码是公开的，我们可以从 %_WINCEROOT%\PUBLIC\COMMON\OAK\DRIVERS\CEDDK\DDK_MAP\ddk_map.c 得到。从 MmMapIoSpace 的实现我们也可以看出 VirtualAlloc 和 VirtualCopy 的用法：

```
PVOID MmMapIoSpace (
IN PHYSICAL_ADDRESS PhysicalAddress,
IN ULONG NumberOfBytes,
IN BOOLEAN CacheEnable
)
{
PVOID pVirtualAddress; ULONGLONG SourcePhys;
ULONG SourceSize; BOOL bSuccess;

SourcePhys = PhysicalAddress.QuadPart & ~(PAGE_SIZE - 1);
SourceSize = NumberOfBytes + (PhysicalAddress.LowPart & (PAGE_SIZE - 1));

pVirtualAddress = VirtualAlloc(0, SourceSize, MEM_RESERVE, PAGE_NOACCESS);
if (pVirtualAddress != NULL)
{
```

```
bSuccess = VirtualCopy(
pVirtualAddress, (PVOID)(SourcePhys >> 8), SourceSize,
PAGE_PHYSICAL | PAGE_READWRITE | (CacheEnable ? 0 : PAGE_NOCACHE));

if (bSuccess) {
(ULONG)pVirtualAddress += PhysicalAddress.LowPart & (PAGE_SIZE - 1);
}
else {
VirtualFree(pVirtualAddress, 0, MEM_RELEASE);
pVirtualAddress = NULL;
}
}
return pVirtualAddress;
}
```

此外，Windows CE 还供了 AllocPhysMem 函数和 FreePhysMem 函数，用来申请和释放一段连续的物理内存。函数可以保证申请的物理内存是连续的，如果函数成功，会返回虚拟内存的句柄和物理内存的起始地址。这对于 DMA 设备尤为有用。在这里就不详细介绍了，读者可以参考 Windows CE 的联机文档。

附录二 Windows CE.NET 高级内存管理

摘要

Microsoft_ Windows CE 的优点之一是它的 Microsoft Win32_ 应用程序编程接口 (API) 支持。无数 Windows 程序员可以利用他们的 Win 32 API 和 MFC 知识相对容易地转移到 Windows CE。Windows CE 可以实现 Win32 API 的子集，但程序员应当永远不要忘记 Windows CE 是与 Windows XP 完全不同的操作系统，Windows CE 具有不同的要求和不同的实现。在设计应用程序或诊断问题时知道 Windows CE 如何实现它的 Win32 兼容性是一件非常重要的事情。

内存管理是 Windows CE 和 Windows XP 之间在实现方面存在很明显差异的地方之一。虽然 Windows CE 支持几乎每个 Win32 内存管理函数（除了我们不赞成对它使用全局堆函数），但是这些内存管理 API 的实现是完全不同的。这些差异可以妨碍那些不熟悉 Windows CE 与 Windows 的桌面版本之间的细微差异的程序员。要了解这些问题位于哪里，必须首先了解 Windows CE 如何管理内存。

系统内存映射

Windows XP 和 Windows CE 都是 32 位操作系统，都同样支持 4 GB 的虚拟地址空间。Windows XP 将地址空间划分为两个 2 GB 区域。上半部地址空间是为系统保留的。下面 2 GB 地址空间则由每个正在运行的应用程序重复使用。

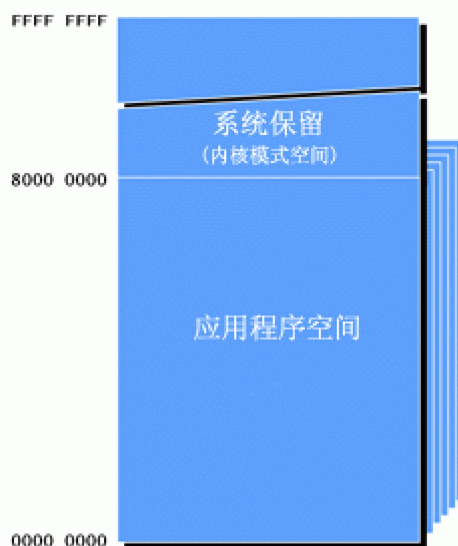


图 1. Windows XP 虚拟内存空间

第 一眼看，Windows CE 的虚拟地址空间是以类似系统保留区域的方式组织的，并且是重复的应用程序空间。图 2 显示了 Windows CE 地址空间。在这里，上部 2GB 地址空间也是为系统保留的。下半部地址空间则划分

为很多区域。该区域的大多数（几乎一半空间）被定义为大型内存区域 (Large Memory Area)。该区域用来分配通常用于内存映射文件的大型内存空间块。

在大型内存区域的下面是另一个大型区域，本文称为*保留区域*。在保留区域的下面，在内存空间的最底端，是一个 64 MB 的区域。该 64 MB 区域，更准确地说是最下面的 32 MB 区域，是每个正在运行的应用程序重复使用的区域。

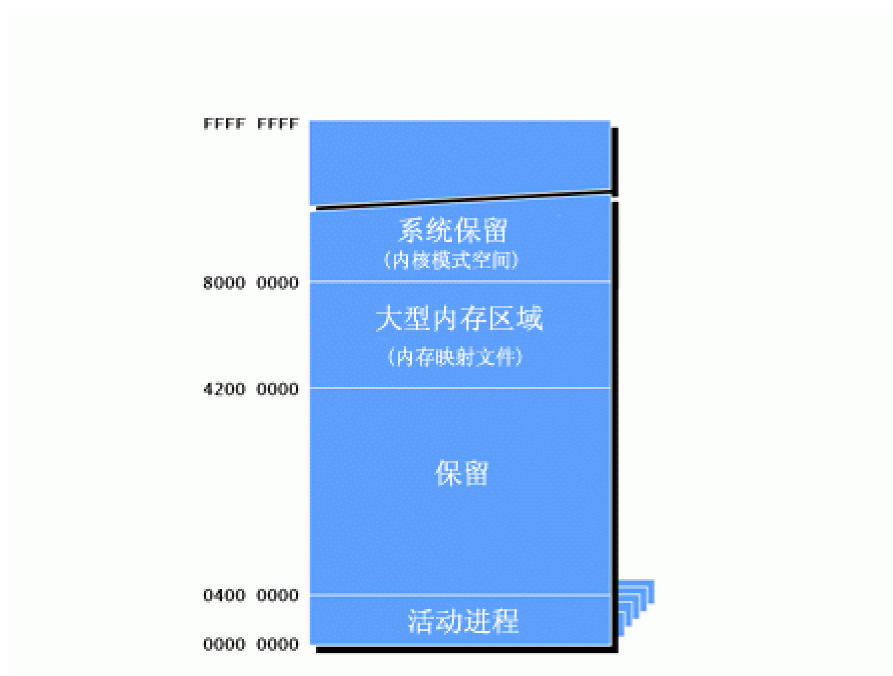


图 2. Windows CE 虚拟内存空间

Windows CE 应用程序内存映射

最 下面的 64 MB 虚拟地址空间是驻留 Windows CE 应用程序的地方。图 3 显示了该应用程序虚拟地址空间。像在 Windows XP 应用程序中一样，应用程序代码从虚拟地址 0x10000 开始加载。应用程序启动时，将在地址空间中为所有代码保留足够的空间。然后，在需要实际代码时，这些代码将被按需分页调度进该地址空间。

在 为代码保留的区域上面，页是为只读和读/写静态数据区域保留的。此外，还为本地堆和应用程序中运行的每个线程的堆栈保留了区域。当线程启动时，为每个堆栈 保留的区域的大小是固定的。只在堆栈增长时才会提交实际的 RAM。另一方面，堆保留了需要在堆中分配 RAM 块时增长的区域。

当加载 “现场执行”(XIP) DLL 时，将从 64 MB 空间的顶部向下加载这些 DLL。当创建 ROM 时，每个 XIP DLL 都会被定址（确定在地址空间中的位置）。加载非 XIP DLL 时，将把它放在 32 MB 边界的下面。非 XIP 的 DLL（也叫作基于 RAM 的 DLL）是指那些从对象存储区加载的 DLL、从 ROM 解压缩的 DLL 或从外部文件系统（例如 Compact Flash 卡）加载的 DLL。应用程序虚拟内存空间中靠上位置的 32 MB 仅用于 XIP DLL。

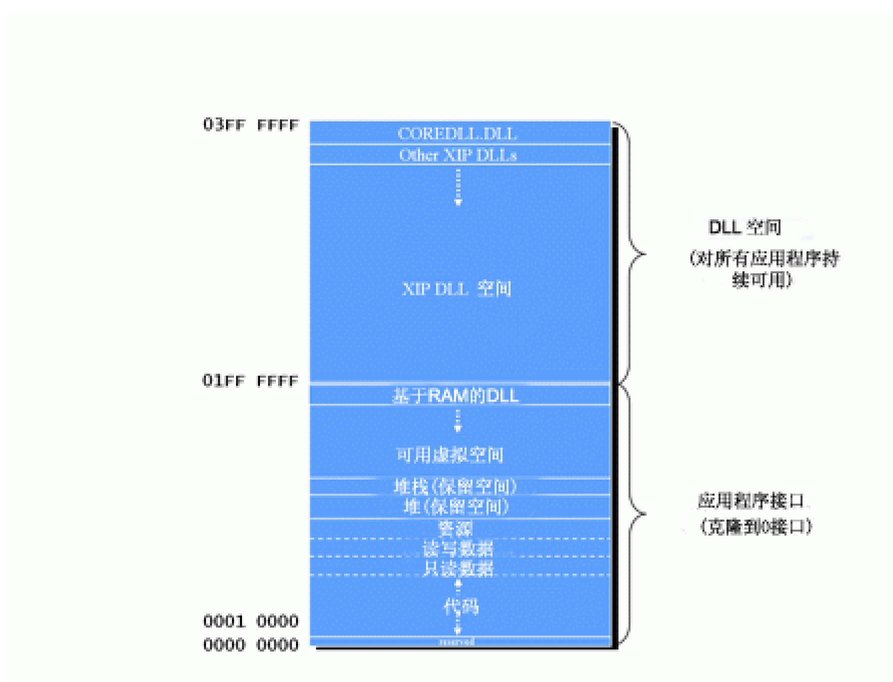


图 3. Windows CE .NET 应用程序虚拟内存空间

由应用程序通过创建单独的堆或直接调用 VirtualAlloc API 所分配的任何其他内存将从底部向上进行分配，分配时，系统将查找第一个足够大、可满足分配的可用区域。

限制因素

尽 管 Windows CE 应用程序的一个限制因素是可用于应用程序的 RAM 数量，但还有另一个主要的限制是应用程序的相对很小的 32 MB 虚拟地址空间。尽管 XIP DLL 被加载在 32 MB 空间的上端，但所有其他内存分配和任何基于 RAM 的 DLL 都必须能够放进应用程序的 32 MB 内存空间中。这个 32 MB 限制“框”不是 Windows CE 程序员所面临的很大问题，因为它是一个可被克服的挑战。

要了解这个看似很大的内存空间有些什么限制性，必须了解 VirtualAlloc API 的操作原理。VirtualAlloc 是任何 Microsoft Win32 操作系统中最基础的内存分配调用。它在页级别分配内存；页是可以被 CPU 分配或释放的最小的内存单位。Windows CE .NET CPU 的页大小是 1024 或 4096 字节，这取决于 CPU。最广泛使用的是 4 KB 页大小。

VirtualAlloc 调用分配内存的过程分为两个步骤。第一步，保留虚拟内存空间的区域。这种保留不会消耗任何 RAM；它只是防止一部分虚拟地址空间被用于其他用途。保留内存空间之后，就可以*提交（commit）*部分或整个区域，这个过程是指将实际物理内存映射到保留区域。VirtualAlloc 函数用于保留内存空间和提交内存。下面显示了 VirtualAlloc 函数的原型。

```
LPVOID VirtualAlloc (LPVOID lpAddress, DWORD dwSize,
                     DWORD flAllocationType,
                     DWORD flProtect);
```

VirtualAlloc 的第一个参数是要分配的内存区域的虚拟地址。使用 VirtualAlloc 提交先前保留的内存块时，使用 lpAddress 参数来标识先前保留的内存块。如果该参数是 NULL，则由系统确定从哪里分配内存区域，并

以 64 KB 为边界。第二个参数是 *dwSize*，它是要分配或保留的区域的大小。因为该参数是以字节而不是页为单位指定的，所以系统会将所请求的大小以下一个页边界为限进行舍入。

flAllocationType 参数指定分配的类型。可以指定以下标志的组合：MEM_COMMIT、MEM_AUTO_COMMIT 和 MEM_RESERVE。MEM_COMMIT 标志用于分配程序使用的内存。MEM_RESERVE 用于保留要随后提交的虚拟地址空间。保留页是无法访问的，直到通过指定区域并使用 MEM_COMMIT 标志进行了另一个 *VirtualAlloc* 调用为止。MEM_AUTO_COMMIT 标志唯一用于 Windows CE 并且很好用，但它不是本文的主题。

因此，要使用 VirtualAlloc 来分配可使用的 RAM，应用程序必须调用 VirtualAlloc 两次，一次保留内存空间，再一次则提交物理 RAM；或者调用 VirtualAlloc 一次，这需要在 *flAllocationType* 参数中组合使用 MEM_RESERVE 和 MEM_COMMIT 标志。

组合保留和提交标志方式所使用的代码更少，并且更快、更简单。该技术通常用在 Windows XP 应用程序中，但用在 Windows CE 应用程序中不是很好。以下代码片段演示了存在的问题。

```
INT i;
PVOID pMem[512];

for (i = 0; i < 512; i++) {
   pMem[i] = VirtualAlloc (0, PAGE_SIZE, MEM_RESERVE | MEM_COMMIT,
                           PAGE_READWRITE);
}
```

该代码片段似乎是无害的。它分配了 512 块内存，每块内存的大小为 1 页。问题是：在 Windows CE 系统上，甚至是在有数兆字节可用 RAM 的系统上，该代码总是会失败。问题在于 Win32 操作系统保留内存区域的方式。

在任何 Win32 操作系统（包括 Windows CE .NET）上，当一个虚拟内存空间区域被保留时，它会将保留区域对齐 64 KB 边界。因而，上面的代码片段试图将保留的 512 个区域的每个区域都对齐 64 KB 边界。Windows CE 应用程序的问题是它们必须位于 32 MB 虚拟内存空间的范围内。在整个应用程序内存空间中该空间的大小只有 512 64 KB，并且它们中的一部分需要用作应用程序代码、本地堆、堆栈和应用程序所加载的每个 DLL 的区域。通常，在对 VirtualAlloc 进行大约 470 次调用之后上面的代码片段将失败。

上述问题的解决方案是首先保留足够用于整个分配的较大区域，然后在需要时提交 RAM，如下所示。

```
INT i;
PVOID pBase, pMem[512];

pBase = VirtualAlloc (0, 512*PAGE_SIZE, MEM_RESERVE, PAGE_READWRITE);

for (i = 0; i < 512; i++) {
   pMem[i] = VirtualAlloc (pBase + (i * PAGE_SIZE), PAGE_SIZE,
                           MEM_COMMIT, PAGE_READWRITE);
}
```

避免该问题的关键是知道这个情况。这只是 Windows CE 应用程序的地址空间中只有 512 个区域所带来的问题影响应用程序的很多地方中的一个。

分配大型内存块

Windows CE .NET 应用程序的地址空间局限于 32 MB 所引起的另一个问题是如何分配大型内存块。当应用程序的整个地址空间被限制在 32 MB 以内时，如果应用程序需要一块 8、16 或 32 Mb RAM 用于具体用途，它怎样才能分配该内存？回答是应用首先用在 Windows CE .NET 早期版本中针对视频驱动程序的一个修复程序。有了它，如果 Windows CE .NET 检测到一个对 **VirtualAlloc** 的调用请求保留超过 2 MB 的地址空间，该地址空间将不会保留在 32 MB 的限制大小中。该内存块将保留在大型内存区域 (Large Memory Area) 中，大型内存区域位于全局内存空间中，正好在 2 GB 系统保留空间的下面。

内存空间已经保留后，应用程序就可以通过调用 VirtualAlloc 来提交在保留空间内的具体页。这就允许应用程序或驱动程序使用大型内存块，即使它存活在 32 MB 的大小的限制内。下面的代码显示了分配 64 MB 块然后提交保留区域的一页的简单情形。

```
PVOID ptrVirt, ptrMem;
ptrVirt = VirtualAlloc (0, 1024 * 1024 * 64, MEM_RESERVE,
                     PAGE_NOACCESS);
if (!ptrVirt) return 0;

ptrMem = VirtualAlloc ((PVOID)((int)ptrVirt+4096),
                       4096, MEM_COMMIT, PAGE_READWRITE);
if (!ptrMem) {
   VirtualFree (ptr, 0, MEM_RELEASE);
   return 0;
}
return ptrMem;
```

前 面的代码还显示了直接处理虚拟内存 API 所具有的一个特性。这就是您可以创建大型稀疏数组，而不会消耗大量 RAM。在上面的代码中，64 MB 保留区域不会消耗任何物理 RAM。在该示例中，唯一被消耗的 RAM 是在第二次调用 VirtualAlloc 以提交页时使用的一个页（4096 字节）。

加载问题

目前，有很多在 Pocket PC 2002 上编程的 Windows CE 程序员。有一个重要问题会影响 Pocket PC 2002 程序员，这个问题与应用程序加载 DLL 有关，尽管对 Windows CE .NET 内存体系结构所作的更改修复了这个问题。要理解该问题，首先必须理解在 Windows CE .NET 如何不同于 Windows CE 3.0 与两个版本的 Windows CE 如何加载和管理 DLL 之间存在的一个主要差异。

Windows CE .NET 的新功能之一是将应用程序的虚拟地址空间从 Windows CE 早期版本的 32 MB 扩展到 64 MB。虚拟空间中可用于 XIP DLL 的高端 32 MB 不可用于 Windows CE 3.0。因此，运行在基于 Windows CE 3.0 的系统上的应用程序必须将它们的 XIP DLL、它们的代码和它们的所有数据加载到 32 MB 虚拟地址空间中。图 4 显示了一个 Windows CE 3.0 应用程序的应用程序内存空间。**图 4** 展示了 Windows CE 3.0 应用程序内存空间的关系图。

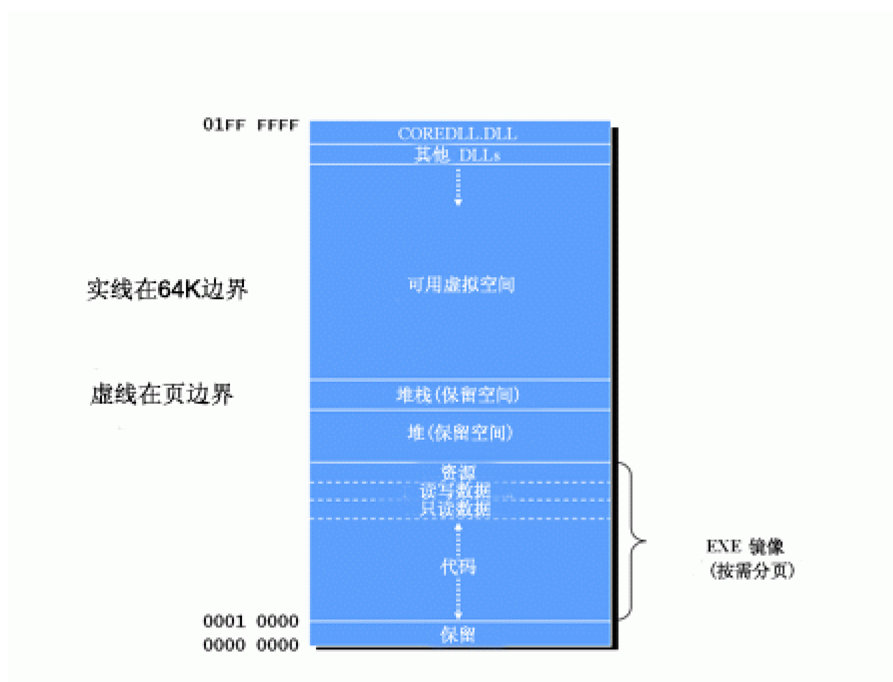


图 4. Windows CE3.0 应用程序虚拟内存空间

因为 Pocket PC 2002 是基于 Windows CE 3.0 的，因此运行在该平台上的应用程序会受到该虚拟内存空间的限制。

DLL 加载

除了在加载 XIP DLL 时 Windows CE 的早期版本与 Windows CE .NET 有额外 32 MB 空间的差异以外，Windows CE 用来加载 DLL 的技术与用于 Windows CE .NET 的技术是相同的。

当 发出加载一个 DLL 的请求时，内核首先检查该 DLL 是否先前已被另一个应用程序加载，如果没有，并且 DLL 不是 XIP DLL，则内核将使用经过修改的从上到下搜索在 32 MB 虚拟内存映射中查找第一个可用的空间。搜索被认为经过修改，是因为内核将避免使用由另一个 DLL 使用的任何地址，即使该 DLL 不是由当前进程加载的。该搜索技术确保了将系统中的每个 DLL 加载在唯一、非重叠的地址中。

之 所以必须使用唯一地址，是因为如果 DLL 是由多个进程加载的，则在所有过程中它必须位于相同的虚拟地址中。通过用唯一的地址加载每个不同的 DLL，内核可以确保如果应用程序想加载由另一个进程先前加载的 DLL，则在其他进程中 DLL 所映射的虚拟地址可用于请求该 DLL 的进程。**图 5** 显示了三个进程分别加载一系列 DLL 的关系图。在该图中，DLL A 由所有三个进程加载在相同地址上。进程 2 加载 DLL C，后者位于比进程 1 所加载的 DLL B 和 DLL A 更低的地址空间中。进程 C 随后加载 DLL A 和它自己的 DLL D。注意，在每个进程中，相同的 DLL 加载在相同的地址中，而每个不同的 DLL 则加载在唯一的地址中。

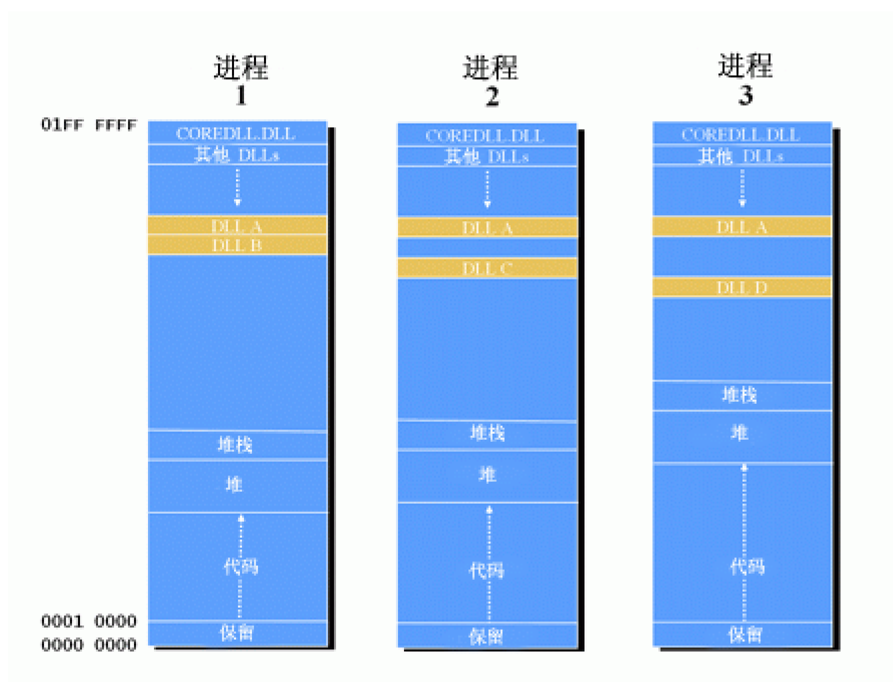


图 5. 加载一系列 DLL 的三个进程

现在考虑如果遇到潜在的问题该怎么办。假设进程 2 加载了很大的 DLL C（如**图 6** 所示）。注意，进程 3 正好是一个大型 .exe 文件，并且在进程 2 已加载了相当大的 DLL C 之后进程 3 也要加载 DLL。很显然，如果进程 3 试图加载还没有被其他进程加载的任何其他 DLL，它很可能遇到麻烦。该示例有点故意设计的成分，因为 DLL C 必须具有难以置信的大小，或者进程 2 必须加载大量 DLL，之后该问题才会自然发生。

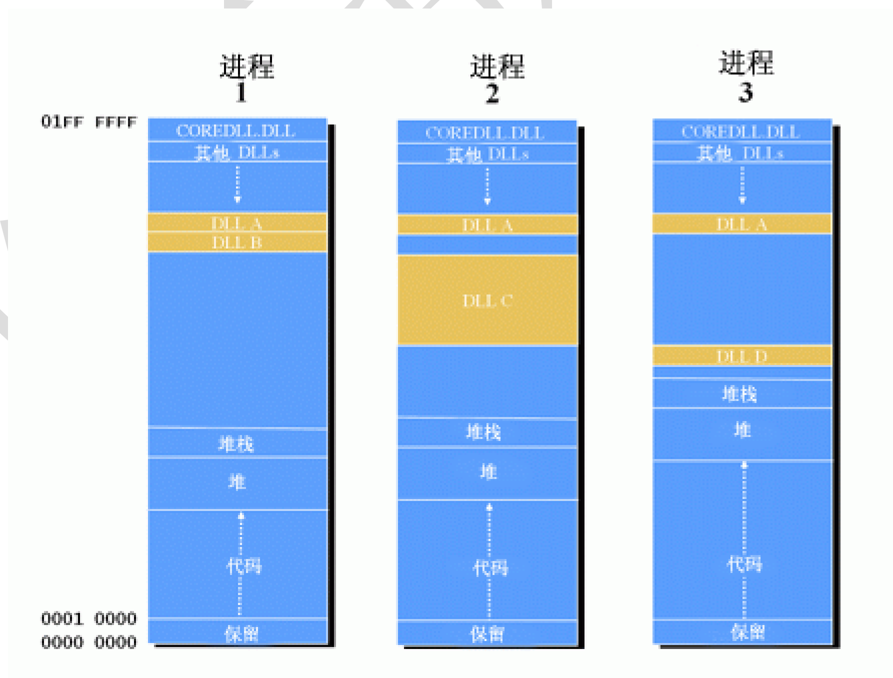


图 6. 三个进程加载一系列 DLL，其中进程 2 加载大型 DLL

讨 论了常规加载 DLL 后，现在是讨论如何处理复杂的 XIP 和非 XIP DLL 的时候了。当 OEM 创建 ROM 镜像时，每个现场执行 DLL 都将被定址在一个唯一地址上。以这种方式，所有 XIP DLL 就能在相互不发生冲突的情况下被加载。因为它们是 XIP，所以包含 DLL 代码的 ROM 可以直接映射到请求它的任何应用程序的虚拟地址空间。XIP DLL 在被进程加载时不能再定址到另一个地址，因为更改基址将涉及修改只读代码。

内核在为非 XIP DLL 查找可用的虚拟地址时，它会从最低定址 XIP DLL 的下面开始搜索可用的虚拟地址。这不是应用程序已加载的最低定址 XIP DLL，而是整个系统中的最低定址 XIP DLL，无论它是否是由任何应用程序加载的。在这里，该技术再次保证了当前加载的每个 DLL 都可以被其他进程加载。尽管该系统运行得很好，但因为某个 DLL 在 Pocket PC 2002 的 Windows CE .NET 中可能只能有唯一的实现，所以有时 DLL 不会被其他进程加载。

在 Pocket PC 2002 上的 Windows CE .NET 实现利用了 Windows CE 3.0 中的功能，该功能允许在设备上使用多个 ROM。该功能允许在系统中使用多个 ROM，即使它们没有连续的物理地址。

上 面已经提到，DLL 需要经过特殊的处理才能成为 XIP。因为对 DLL 的定址需要更改 DLL 的代码，所以在创建 ROM 镜像时 DLL 必须被定址。创建第一个 ROM 时，ROM 创建工具将对每个 DLL 定址，以便它不会与 ROM 中的任何其他 DLL 发生重叠。

使 用多个 XIP 区域意味着 DLL 加载问题需要内核设计者重新考虑。要确保在多个 XIP 区域系统上 XIP DLL 永远不会重叠，必须将第二个 ROM 上的 DLL 定址到比第一个 ROM 镜像的最低 DLL 更低的虚拟地址。如果使用了其他 ROM，则这些 XIP 区域中的 DLL 还必须定址到比前一个 ROM 更低的地址。

由于其他原因，使用多个 ROM 镜像很容易。如果 OEM 或 Microsoft 想更新 Windows CE 镜像的一部分，它们可以为具体 ROM 发出更新，而不必更新整个系统。为了保证一个 ROM 的更新不需要有对另一个 ROM 的更改，Microsoft 鼓励不要将定址于较低镜像中的 DLL 定址到前一个镜像中最低 DLL 的地址，而应当定址在比它更低的地址上，以在一组 DLL 和另一组 DLL 之间人为引入虚拟内存空隙。

负责 Pocket PC 2002（基于 Windows CE 3.0）的 Microsoft 内部开发人员最大限度地利用了多个 XIP 区域。大多数 Pocket PC 实现都有五个或更多个 XIP 区域。问题是区域之间的空隙太大。Pocket PC 2002 镜像中的最低定址 XIP DLL 通常定址在 0x0100000 以下。因为 Windows CE 将基于 RAM 的 DLL 放在最低 XIP DLL 的下面，所以可供基于 RAM 的 DLL、应用程序代码、它的堆和堆栈使用的空间没有限制在 32 MB 虚拟地址空间的范围内，而是在最低 XIP DLL 下面的空间中（小于 16 MB）。

图 7 显示了 Pocket PC 2002 的问题。注意，XIP DLL 的虚拟内存空间中的区域相当大。事实上，这幅图很保守，因为它没有显示 XIP 区域接管虚拟内存空间的一半的情形，而 Pocket PC 2002 上通常就是这种情况。注意基于 RAM 的 DLL 的加载；A、B、C 和 D 位于虚拟地址空间中低很多的位置。

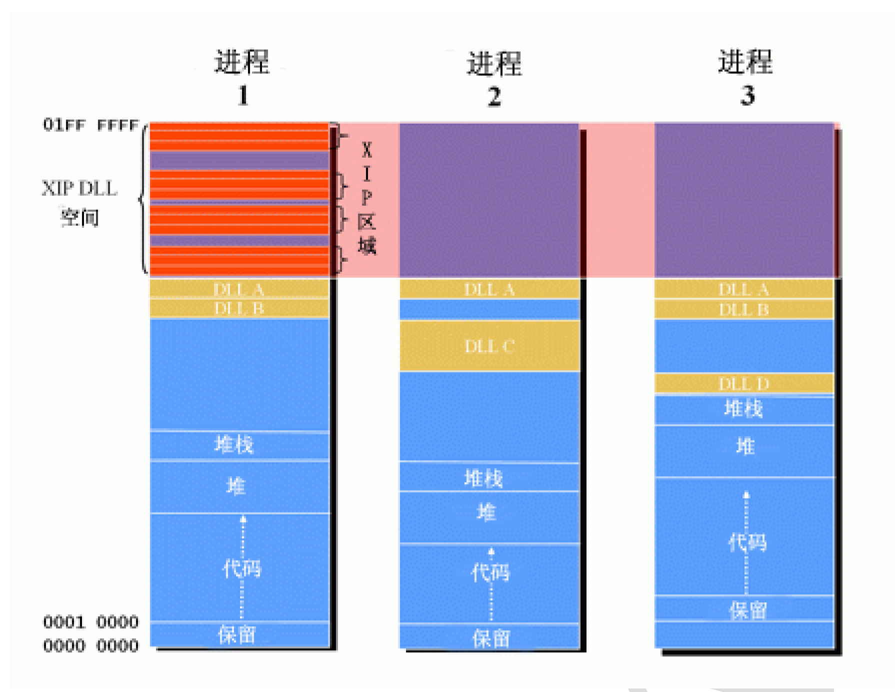


图 7. 在 Pocket PC 2002 上加载 DLL，大部分虚拟地址空间被 XIP DLL 使用

对 于处理海量数据的公司应用程序，公司开发人员被迫在他们的 Windows CE 应用程序中使用大型数据库。通常数据库引擎被实现为 DLL，而它通常很大。在上面的示例中，数据库 DLL 是制造麻烦的 DLL C。可用于 Pocket PC 2002 应用程序的虚拟内存空间小于 16 MB，而人们又需要大型的、基于 RAM 的 DLL，这使得很多开发人员发现他们的应用程序将由于缺少空间而无法运行 — 不是缺少 RAM，而是虚拟内存空间。

组合 DLL

可 用来减轻 Pocket PC 2002 上的该问题的技术有不少。首先，开发人员应当通过将小型 DLL 组合成更大的 DLL 来减少 DLL 的数目。每个 DLL 至少占据一个 64 KB 区域。如果应用程序有 4 个 DLL，每一个的大小是 20 KB，则 DLL 使用的总计内存空间是 256 KB。通过组合四个 DLL，所得到的大型 DLL 将仅消耗 64 KB 虚拟内存空间 — 代码只占用 60 KB，但最低内存使用量是 64 KB。常规规则是，将 DLL 组合成（但不超过）64 KB 的倍数的大小。在某些包含过多小型 DLL 的应用程序中，只需将 DLL 组合成几个大型 DLL 就能解决应用程序的 DLL 加载问题。

将 DLL 代码转移到应用程序

在 Pocket PC 2002 中减少 DLL 问题的另一个方法是将 DLL 中的代码转移到应用程序。即使多个进程共享代码，有时在多个进程中复制代码也是有利的，因为不同进程将独立于其他应用程序被加载到内存中。

首 先，将代码移动到应用程序中似乎没有帮助 — 代码仍然在应用程序的 32 MB 虚拟空间中。但是，这里的关键是要使某些代码成为不需要大型的、基于 RAM 的 DLL 的大型应用程序，而使其他代码成为加载和使用基于 RAM 的 DLL 的小型应用程序。在该技术中，大型应用程序执行大多数业务逻辑和用于加载大型 DLL 的小型应用程序。如果大型应用程序需要得到大型 DLL 的服务，它必须使用进程间通信让较小的进程调用该 DLL，并通过再次使用进程间通信将数据返回给大型进程。

定义 DLL 加载顺序

减 少 DLL 数或转移应用程序的代码还不够，下面讨论更基本的方法：手动指定 DLL 的加载顺序。加载顺序是重要的，因为如果大型 DLL 在早期加载，它将迫使所有随后的小型 DLL 的加载地址向下转移。通常，大型 DLL 被单个应用程序使用。但如果它被早期加载，它可以迫使其他应用程序 DLL 的加载地址向下转移到无法加载这些 DLL 的位置，从而冲击其他应用程序。

解决方案是首先加载小型 DLL，然后让会造成影响的大型 DLL 在晚期加载，甚至最后加载。这就产生了如何强制执行 DLL 加载顺序的问题。一个方式是对应用程序套件中不同进程的启动顺序进行排队，但这有时会有问题。

另 一个定义 DLL 加载顺序的方式是编写一个运行于主要应用程序之前的小型应用程序，让它通过重复调用 Win32 函数 LoadLibrary，按定义好的顺序加载基于 RAM 的 DLL。DLL 加载程序在主应用程序的生存期内一直运行，然后终止。它甚至可以通过调用 CreateProcess 来启动主应用程序，并通过阻塞 CreateProcess 所返回的进程句柄而进入等待状态，直到主应用程序终止。加载 DLL 的应用程序不会使用很多 RAM，因为被加载的 DLL 最后全部都要由其他进程来加载。

为解决 Pocket PC 2002 上的 DLL 加载问题而讨论的所有解决方案都各有缺点。没有一个是完美的或不可辩驳的。但是，它们确实是开发人员用来开发其产品的解决方案。以后发布的 Pocket PC 应当解决该问题，但对开发 Pocket PC 2002 产品的开发人员来说，及时解决问题是关键的。

通过知道 Windows CE 如何管理内存，开发人员就能更快地避免缺陷，并诊断问题。理解 Windows CE 如何管理 DLL 将有助于避免 Pocket PC 2002 应用程序中潜在的问题。即使未来发布的 Pocket PC 解决了该问题，已经在使用的数百万设备仍然需要应用程序。知道从哪里查找问题是找到并解决问题的第一步。

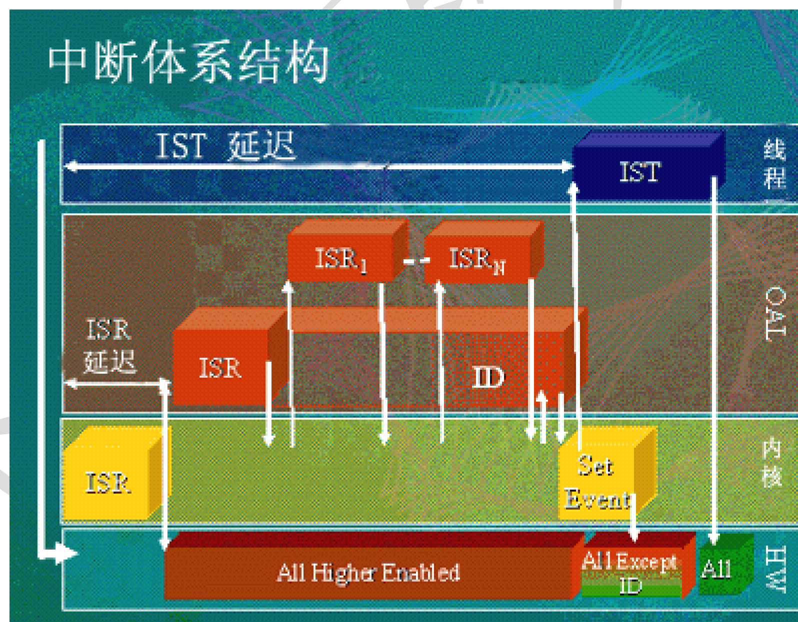
附录三 Windows CE.NET 中的中断体系结构

概述

通过 Microsoft Windows CE .NET，Microsoft 已经升级了 Windows CE 的中断体系结构。该操作系统 (OS) 所具有的处理共享中断的能力极大地扩展了 Windows CE .NET 支持许多中断体系结构的能力。本文从原始设备制造商 (OEM) 和应用程序开发人员的角度探讨了处理中断的方案。本文还探讨了 OEM 适配层 (OAL) 中断服务例程 (ISR) 处理；提供了可安装 ISR，包括一个简单的入门级外壳程序；介绍了中断服务线程 (IST) 中断处理，并提供了一个初始化和执行模板。最后，本文分析了 ISR 和 IST 的延迟根源。

中断体系结构

探讨 Microsoft Windows CE .NET 中断体系结构的第一步是定义中断过程中硬件、内核、OAL 和线程交互的总体模型。下图大概说明了这些不同级别的职责以及导致状态变化的转换。



该图阐述了中断过程中的主要转换，时间按从左到右的顺序递增。该图的最低层为硬件和中断控制器的状态。次低层是中断服务过程中的内核交互。OAL 描述了主板支持软件包 (BSP) 的职责。最顶层阐述了中断服务所需的应用程序或驱动程序线程交互。该图阐述了单个中断过程中的交互；它表示了 Windows CE .NET 拥有共享中断的新能力。

活动从该图最左侧部分以直线表示的中断开始。生成了一个异常，导致内核 ISR 向量被加载到处理器中。内核 ISR 与硬件交互，禁用所有处理器上的所有具有相同和较低优先级的中断（ARM 和 Strong ARM 体系结构除外）。然后，内核推进到已为该特定中断注册的 OAL ISR。此后，OAL ISR 既可以直接处理中断，也可以使用 NKCallIntChain 遍历已安装的 ISR 列表。主 ISR 或任何已安装的 ISR 随后执行任意工作，并且为该设备返回名为 SYSINTR 的映射中断。如果该 ISR 确定其相关设备没有导致该中断，该 ISR 将返回 SYSINTR_CHAIN，这会使 *NKCallIntChain( )* 遍历 ISR 列表以到达链中的下一个中断。ISR 按照它们的安装顺序调用（它们在安装时会在调用列表上创建一个优先级）。

在调用了单个 ISR 或其相关 ISR 链之后，返回值可能为下列值之一：

返回值	**操作**
SYSINTR_NOP	中断不与设备的任何已注册 ISR 关联。内核启用所有其他中断。
SYSINTR	中断与已知的已注册 ISR 和设备关联。
SYSINTR_RESCHED	中断是由请求 OS 重新调度的计时器到期引起的。

SYSINTR 返回值是我们讨论的重点。一旦 ISR 完成，内核将重新启用处理器上除已识别的中断之外的所有中断。然后，内核将通知与 SYSINTR 值关联的事件。

然后，驱动程序或应用程序的 IST 将能够运行（假设它是准备好运行的最高优先级线程）。IST 将与相关设备通讯，并从完成它的中断交互的设备中读取所有必要的数据。然后，IST 用关联的 SYSINTR 值来调用 *InterruptDone( )*，以通知它已完成。

内核在接收到 SYSINTR 值的 InterruptDone 时，将重新启用指定的中断。只有从这时开始，才能接收该设备的其他中断。

这只是对 Windows CE .NET 内部活动的中断序列的一个粗略介绍。现在，我们将详细研究上述每个组件及其职责。

OAL ISR 处理

OAL ISR 是属于平台的基本中断处理程序。下面是 X86 平台的实际 ISR。配置分析和 ILTiming 支持已被删除。X86 ISR 是所有基于 Windows CE 的平台的代表。它演示了能够处理系统中所有中断的单个 ISR。

该 ISR 的目标是向内核交还引起中断的相关设备的 SYSINTR 号。ISR 执行以下活动序列。

- 从 **PICGetCurrentInterrupt** (PIC) 中获取当前硬件中断
- 如果该中断是 **INTR_TIMER0**（系统计时器）
  - 更新 OS 的 **CurMSec** 保持时间
  - 检查并确认是否已经注册了重新启动地址 (**RebootHandler**)
- 如果中断是 **INTR_RTC**
  - ISR 检查并确认闹钟是否已到期 (**SYSINTR_RTC_ALARM**)
- 如果中断小于 **INTR_MAXIMUM**
  - 调用中断链 (**NKCallIntrChain**)
  - 将 **NKCallIntrChain** 的返回值设置为该返回值
  - 如果中断链未包含中断：(**SYSINTR_CHAIN**)

  映射当前硬件中断 (**OEMTranslateIRQ**)

  如果该中断被注册到 **OEMInit** 中的 **HookInterrupt**

  从 **OEMTranslateIRQ** 返回 **SYINTR** 值

  如果该中断未注册，则返回 **SYSINTR_NOP**

  - 启用除当前中断以外的所有中断。(**PICEnableInterrupt**}
  - 完成恰当的中断结束工作以通知 **PIC** 中断已完成 (**EOI**)
- ISR 返回下列值之一：
  - **SYSINTR_NOP** — 没有任何 ISR 包含该中断
  - **SYSINTR_RESCHED** — 重新调度计时器已到期
  - **SYSINTR** — ISR 已经包含该中断
  - **SYSINTR_RTC_ALARM** — 闹钟已到期

```
ULONG PeRPISR(void)
```

```
{
ULONG ulRet = SYSINTR_NOP;
UCHAR ucCurrentInterrupt;

ucCurrentInterrupt = PICGetCurrentInterrupt();

if (ucCurrentInterrupt == INTR_TIMER0) {

            CurMSec += SYSTEM_TICK_MS;
            CurTicks.QuadPart += TIMER_COUNT;

            if ((int) (CurMSec - dwReschedTime) >= 0)
                ulRet = SYSINTR_RESCHED;
        }

        //
        // Check if a reboot was requested.
        //
        if (dwRebootAddress) {
          RebootHandler();
        }

   } else if (ucCurrentInterrupt == INTR_RTC) {
          UCHAR cStatusC;
         // Check to see if this was an alarm interrupt
          cStatusC = CMOS_Read( RTC_STATUS_C);
          if((cStatusC & (RTC_SRC_IRQ)) == (RTC_SRC_IRQ))
                ulRet = SYSINTR_RTC_ALARM;
      } else if (ucCurrentInterrupt <= INTR_MAXIMUM) {
          // We have a physical interrupt ID, return a SYSINTR_ID
```

```
        // Call interrupt chain to see if any installed ISRs handle this
// interrupt
        ulRet = NKCallIntChain(ucCurrentInterrupt);

        if (ulRet == SYSINTR_CHAIN) {
          ulRet = OEMTranslateIrq(ucCurrentInterrupt);
         if (ulRet != -1)
           PICEnableInterrupt(ucCurrentInterrupt, FALSE);
         else
            ulRet = SYSINTR_NOP;
        } else {
          PICEnableInterrupt(ucCurrentInterrupt, FALSE);
        }
    }

    if (ucCurrentInterrupt > 7 || ucCurrentInterrupt == -2) {
        __asm {
              mov al, 020h    ; Nonspecific EOI
              out 0A0h, al
        }
    }
   __asm {
        mov al, 020h        ; Nonspecific EOI
        out 020h, al
    }

    return ulRet;
}
```

如果 ISR 没有为已经用 OAL 的 OEMInit 中的 HookInterrupt 初始化的中断安装，则该 ISR 将返回适当的 SYSINTR 值。

注 如果只能通过 IST 交互为设备提供服务，则不需要为中断安装可安装的 ISR。通过对 **OAL** 的 **OEMInit** 中的 **HookInterrupt** 进行调用以启用中断就足够了。

ISR 代码是一段非常小且快速的代码。它的执行时间将直接影响整个系统中的中断的延迟。Windows CE 3.0 中引入的中断体系结构更改是能够**嵌套中断**。在进入 OAL ISR 的那一刻，所有具有较高优先级的中断都已被启用。ISR 可能被占先。如果该 ISR 内部的计时非常关键，则可能要求在该时间段内禁用中断。就像 ISR 执行时间一样，中断被关闭的这一时间将增加平台的最差情形延迟。

当 ISR 交还与特定设备相关的 SYSINTR 时，内核将通知 IST 醒来。处理驱动程序或应用程序内部代码的 IST 中断负责结束中断交互。

可安装的 ISR

可安装的 ISR 是为响应 Windows CE .NET 为嵌入式空间带来的开放性而创建的。OEM 再也不必完全负责平台和应用程序代码了。现在平台提供商和应用程序开发人员都可涉及嵌入式空间这一领域的工作。如果某个应用程序开发人员在使用 Windows CE 3.0 的平台上向开放总线添加了新的设备，OEM 将必须说服该 OEM 将 ISR 添加到该平台。

要将 ISR 安装到平台中，需要完成两个步骤：

- 调用 LoadIntChainHandler 函数以加载包含 ISR 代码的 DLL。
- 必须将 ISR 编码为用 SYSINTR_ . . . 响应进行响应，就像在 OAL ISR 中一样。

LoadIntChainHandler 函数将 ISR 动态链接库 (DLL) 加载到内核的地址空间中。这意味着代码不能调用任何非内核函数，包括任何 C 语言运行时库函数。记住，某些结构到结构赋值会降格为 memcpy 调用，必须检查所有代码以确保不需要任何外部库（即使这些库是由编译器创建的）。

下面的源代码示例说明了一个用于创建可安装的 ISR 的基本外壳程序。有四个函数：

- **DLLEntry** — 接收进程和线程附加消息
- **InfoCopy** — 在进行任何结构赋值时使用的复制例程
- **IOControl** — 任何使用 **KernelLibIOControl** 的 IST 调用的处理程序
- **ISRHandler** — 实际的 ISR

```
BOOL __stdcall  DllEntry(   HINSTANCE hinstDll,
```

```
        DWORD dwReason,
        LPVOID lpReserved )
{
     if (dwReason == DLL_PROCESS_ATTACH) {}

     if (dwReason == DLL_PROCESS_DETACH) {}

     return TRUE;
}

// The compiler generates a call to memcpy() for assignments of large
  objects.
// Since this library is not linked to the CRT, define our own copy
  routine.
void  InfoCopy( PVOID  dst, PVOID  src, DWORD size  )
{
    while (size--) {
*((PBYTE)dst)++ = *((PBYTE)src)++;
}
}

BOOL IOControl(     DWORD   InstanceIndex,
         DWORD   IoControlCode,
          LPVOID  pInBuf,
          DWORD   InBufSize,
         LPVOID  pOutBuf,
          DWORD   OutBufSize,
          LPDWORD pBytesReturned )
{
switch (IoControlCode) {
```

```
    case IOCTL_DEMO_DRIVER:
            // Your I/O Code Here
            return TRUE;
            break;

    default:
            // Invalid IOCTL
            return FALSE;
}

return TRUE;
}

DWORD   ISRHandler(  DWORD InstanceIndex  )
{
  BYTE  Value;

Value = READ_PORT_UCHAR((PUCHAR)IntrAddress );

// If interrupt bit set, return corresponding SYSINTR
if ( Value & 0x01 )
{
  return SYSINTR_DEMO;
}
else
{
  return SYSINTR_CHAIN;
}
}
```

ISR 处理程序代码使用端口 I/O 调用来检查设备的状态。您的方案可能要求复杂得多的询问。如果该设备不是中断源，则返回值将是 SYSINTR_CHAIN。此返回值告诉 NKChainIntr 函数该设备不是中断源，

应该评估链中的其他 ISR。如果 ISR 返回有效的 SYSINTR，则 NKChainIntr 将立即返回并且不调用列表中的任何其他 ISR。这将提供优先级排序。第一个加载的可安装 ISR 被首先加载到该列表中（或具有最高优先级），然后将后续可安装 ISR 添加到该列表的底部。由于优先级和执行速度这两方面的原因，应该首先安装链中具有最高优先级的可安装 ISR。

IST 中断处理

处理来自应用程序或驱动程序的中断需要进行两个步骤的处理。首先，必须使用关联的事件初始化中断。其次，IST 必须等待中断事件以响应内核中的中断。

中断初始化

以下示例代码将设置 IST 并将 IST 与特定的中断相关联。初始化中断的关键步骤包括：

- 创建事件
- 获取 IRO 的系统中断号
- 创建**挂起的**中断线程 (IST)
- 调用 **InterruptInitialize** 以创建 IRQ 与事件之间的关联
  - 创建未挂起的 IST 可能会导致 **InterruptInitialize** 失败，因为该事件已经处于被等待状态
- 将线程优先级设置为相应的优先级
- 恢复 IST

```
Void SetupInterrupt( void )
{
// Create an event
//
g_hevInterrupt = CreateEvent(NULL, FALSE, FALSE, NULL);
if (g_hevInterrupt == NULL)
{
    RETAILMSG(1, (TEXT("DEMO: Event creation failed!!!\r\n")));
```

```
    return;
}

// Have the OAL Translate the IRQ to a system irq
//
fRetVal     = KernelIoControl( IOCTL_HAL_TRANSLATE_IRQ,
                &dwIrq,
                sizeof( dwIrq ),
                &g_dwSysInt,
                sizeof( g_dwSysInt ),
                NULL );

// Create a thread that waits for signaling
//
g_fRun   = TRUE;
g_htIST       = CreateThread(NULL,   // Security
          0,      // No Stack Size
          ThreadIST,  // Interrupt Thread               NULL,    // No
  Parameters
          CREATE_SUSPENDED, // Create Suspended
          &dwThreadID  // Thread Id
          );

// Set the thread priority - arbitrarily 5
//
m_nISTPriority = 5;
if( !CeSetThreadPriority( g_htIST, m_nISTPriority ))
{
    RETAILMSG(1,(TEXT("DEMO: Failed setting Thread Priority.\r\n")));
    return;
}
```

```
// Initialize the interrupt
//
if ( !InterruptInitialize(g_dwSysInt,g_hevInterrupt,NULL,0) )
{
    RETAILMSG (1, (TEXT("DEMO: InterruptInitialize failed!!!\r\n")));
    return;
}

// Get the thread started
//
ResumeThread( g_htIST );

}
```

需要注意的是，对 InterruptInitialize 的调用仅采用 SYSINTR 值和事件作为参数。内核不知道或者不关心将要等待该事件的线程。这样，就可以建立多种应用程序和驱动程序体系结构。应用程序的简单主循环可以初始化中断，然后立即等待该事件。中断只能与一个事件关联，并且该事件不能用于对 WaitForMultipleObjects 的调用中。我们将观察一个简单的为中断提供服务的线程。这是大多数实现中的标准解决方案。

IST － 中断服务例程

本节提供了一个 IST 的示例代码。该 IST 中断处理线程的关键组件包括：

一　等待中断事件。

二　确认有一个来自 OS 的脉动性事件

三　执行任何必要的板级中断处理以完成中断。在该示例中，我们将确认该中断。

四　在尽可能短的时间内处理该中断

五　创建 CELOGDATA 以供在 Kernel Tracker 中查看。

六　检查并确认是否设置了 **g_fPRRunning** 标志，然后设置 **g_hevPRStart** 事件。

七　调用 InterruptDone()。

7.1　在调用 InterruptDone 之前，OS 不会提供此 IRQ 上的其他中断。

八　再次等待中断事件。

```
DWORD  WINAPI  ThreadIST( LPVOID lpvParam )
{
  DWORD  dwStatus;
  BOOL  fState = TRUE;

  // Always chec the running flag
  //
  while( g_fRun )
  {
    dwStatus  = WaitForSingleObject(g_hevInterrupt, INFINITE);

    // Check to see if we are finished
    //
    if(!g_fRun ) return 0;

    // Make sure we have the object
    //
    if( dwStatus == WAIT_OBJECT_0 )
    {

      // Do all interrupt processing to complete the interaction
      // with the board so we can receive another interrupt.
      //
      if (!( READ_REGISTER_ULONG(g_pBoard Register) & INTR_MASK))
      {
        RETAILMSG(1, (TEXT("DEMO: Interrupt..."));
        g_dwInterruptCount ++;
```

```
      }

      // Finish the interrupt
      //
      InterruptDone( g_dwSysInt );

    }
  }

  return 0;
}
```

该示例读取一个 ULONG 寄存器以确定中断状态。您只需用您的代码替换该代码段。非常关键的一点是，要使 IST 处理尽可能地简单。如果将来需要处理来自该设备的数据：

- 在 IST 中尽可能快速地从该设备获取数据。
- 创建一个事件，以通知某个优先级较低的线程完成该工作。
- 通过 InterruptDone 从该 IST 中立即返回。
- 让优先级较低的线程进一步处理数据。
- 在 IST 与优先级较低的线程之间放置 FIFO 以处理溢出。

导致延迟的因素

从 Windows CE .NET 中的中断体系结构示意图中，可以了解硬件、内核、OAL 与驱动程序/应用程序线程之间的交互。Microsoft 已经提供了多种工具（包括 ILTiming、CEBench 和 Kernel Tracker），以便帮助您评估平台上的 Windows CE .NET 的性能。通过了解导致 ISR 和 IST 延迟的因素，有助于确定调查领域。

ISR 延迟

正如您在本文前面的**中断体系结构示意图**中可以看到的，ISR 延迟被定义为从发生中断到 OAL ISR 首次执行之间的时间。因为当中断被关闭时，中断不会在处理器中引发异常，所以第一个导致延迟的因素是

系统中的中断被关闭的总时间。在每个机器指令开始执行时都将检查是否有处理器中断。如果调用了长字符串移动指令，则会锁定中断，从而造成第二个延迟源，即总线访问锁定处理器的时间量。第三个因素是内核导向 OAL ISR 处理程序所花费的时间量。这是一个进程上下文切换。总之，导致 ISR 延迟的因素包括：

- 中断被关闭的时间。
- 总线指令锁定处理器的时间。
- 内核 ISR 的执行时间加上导向 OAL ISR 的时间。

IST 延迟

本文前面的体系结构示意图中显示，IST 延迟是从中断发生到执行 IST 中的第一行代码之间的时间量。这与 Windows CE .NET 中的 Microsoft 度量工具的输出不同。Microsoft 工具将 IST 延迟定义为从 OAL ISR 执行结束到 IST 开始之间的时间。因为标准的 ISR 花费的时间很少，您需要将 ISR 延迟和 Microsoft 度量工具所得到的 IST 延迟加起来，才能获得"中断体系结构示意图"中所定义的 IST 延迟。

导致 IST 延迟的第一个因素是本文前面定义的 ISR 延迟。第二个因素是 ISR 执行时间。根据共享中断调用链的长度的不同，此时间是可变的。对于延迟较小的情况，没有必要对永远不会被共享的中断调用 **NKCallIntChain**。

Windows CE 中的内核函数（如计划程序）被称为 KCALL。在这些 KCALL 执行期间，将设置一个软件标志，以便让计划程序知道它此时不能被中断。仍然将调用 ISR，但用于重新调度 OS 或调度 IST 的返回值将被延迟，直至 KCALL 完成为止。这一不可占先的时间是导致 IST 延迟的第三个因素。最后，内核必须调度 IST。这一上下文切换是导致延迟的最后一个因素。总之，导致 IST 延迟的因素包括：

- ISR 延迟时间
- OAL ISR 执行时间
- OS 执行 KCALL 的时间
- 调度 IST 的时间

小结

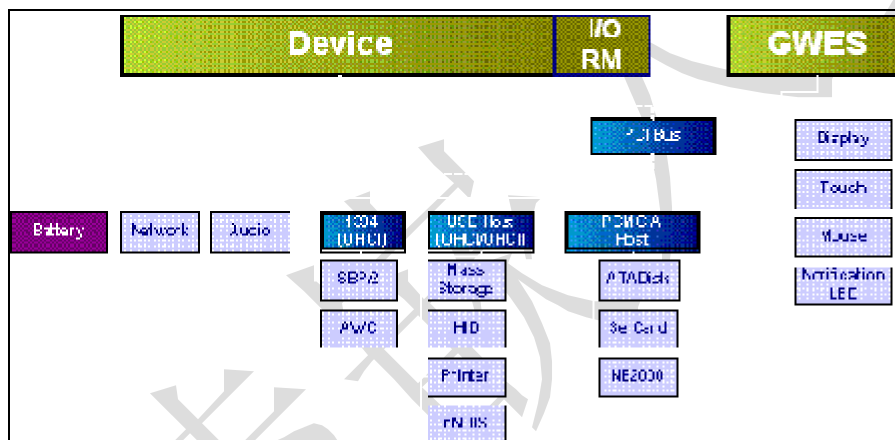
通过 Windows CE .NET，Microsoft 已经升级了 Windows CE 中断体系结构。该 OS 所具有的处理共享中断的能力极大地扩展了 Windows CE .NET 支持许多中断体系结构的能力。这一中断体系结构方面的知识可以大大加快调查驱动程序和延迟问题的速度。操作系统交互模型是了解该体系结构的关键。共享中断已经大大提高了 Windows CE .NET 的开放性，能够支持遍布于不同公司之间以及公司内部的平台提供商和应用程序开发人员方案。了解延迟根源将有助于诊断驱动程序和实时问题。Windows CE .NET 中的中断结构定义完善且易于理解。简而言之，"它不是魔术！"

附录四 如何在 Windows CE 5.0 中开发和测试设备驱动程序

第一部分：建立设备驱动程序

在本练习中，您将使用 Platform Builder 来添加作为设备驱动程序的项目。

在 开始编写驱动程序之前，您应该了解设备驱动程序的用途。驱动程序将基础硬件从操作系统中抽象出来，使之更好地面对应用程序开发人员。应用程序开发人员无需 知道显示硬件或串行硬件的详细信息 — 例如，串行设备是用 Universal Asynchronous Receiver/Transmitter (UART) 实现的还是用 field-programmable gate array (FPGA) 实现的。在大多数情况下，应用程序开发人员根本不需要知道硬件是如何实现的。



Microsoft Windows 为开发人员公开了调用硬件的应用程序编程接口 (API)，他们不需要知道物理硬件的情况。例如，为了向串行端口写入数据，应用程序开发人员只需调用 COMx 上的 CreateFile( )（其中 x 表示您要打开的串行端口编号，例如 COM1 代表串行端口 1），再调用 WriteFile( ) 以将一些字节数据写入串行端口，然后调用 CloseHandle( ) 以关闭串行端口。不管基础串行硬件是什么（也不管您运行的是哪个 Windows 操作系统），API 都会以同样的顺序执行。

相同的情况也适用于其他 API：如果您希望在显示表面画一条线，那么您只需调用 PolyLine( )、MoveToEx( ) 或 LineTo( )。作为应用程序开发人员，大多数情况下您都不需要知道显示硬件的情况。此处调用的 API 将返回显示表面的维数、颜色深度等等。

好 消息是开发人员可以调用一个一致的、众所周知的 API 集。这些 API 将他们的应用程序从基础硬件中抽象出来。这至关重要，因为应用程序开发人员无法知道应用程序是运行在便携式计算机上，还是运行在 Tablet PC 上，抑或运行在桌面计算机上。无论电脑以 1024×768 还是 1600×1200 的分辨率运行，应用程序开发人员都可以在运行时查询屏幕分辨率和颜色深度，因此不需要构建只在特定硬件上运行的应用程序。

驱动程序只是一个 动态链接库(DLL)。将 DLL 加载到父进程地址空间；然后父进程就可以调用从该 DLL 公开的任何接口。通常，父进程通过调用 LoadLibrary( ) 或 LoadDriver( ) 来加载驱动程序。LoadDriver 不仅将 DLL 加载到父进程地址空间中，而且还要确保 DLL 没有“paged out”。

调用进程如何知道从您的 DLL 或驱动程序公开了哪些 API 或函数呢？父进程调用 GetProcAddress( )，后者可以获取函数名称和所加载的 DLL 的 hInstance。如果函数存在，调用返回该函数指针；如果没有从 DLL 公开该函数，则返回 NULL。

流驱动程序也公开了一个众所周知的函数集。对于流驱动程序，您会希望能够将字节流写入设备中，或者从设备中读取字节流。因此，在前面使用的串行端口示例中，您可能希望从您的驱动程序公开如下函数集：**Open**、**Close**、**Read** 和 **Write**。流驱动程序还公开一些其他函数：**PowerUp**、**PowerDown**、**IOControl**、**Init** 和 **DeInit**。

您可以将现有的操作系统映像用于模拟器平台（Basic Lab MyPlatform 平台最理想）。然后，您就可以将 DLL/驱动程序项目添加到该平台了。

在构建并下载了该平台之后（这表明操作系统启动并运行良好），您需要创建您的主干驱动程序。您可以使用 **File** 菜单上的 Platform Builder **New Project or File** 命令创建一个 Microsoft Windows CE DLL。创建用于公开函数或资源的 DLL 与创建用作驱动程序的 DLL 之间没有什么不同；唯一的不同之处在于 DLL 公开哪些函数，以及如何在平台上注册或使用 DLL。

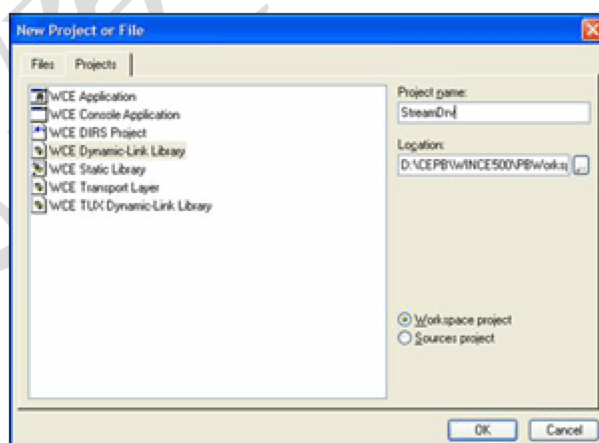
此 外，一种创建国际化应用程序的方法是，首先创建包含一组核心语言字符串、对话框和资源的基本应用程序，然后创建许多外部 DLL，其中每个都包含针对特定区域设置的对话框、字符串和资源。然后，应用程序就可以在运行时加载相应的语言资源。只需要添加 DLL 文件，您就可以将语言添加到应用程序中。

添加一个作为设备驱动程序的项目

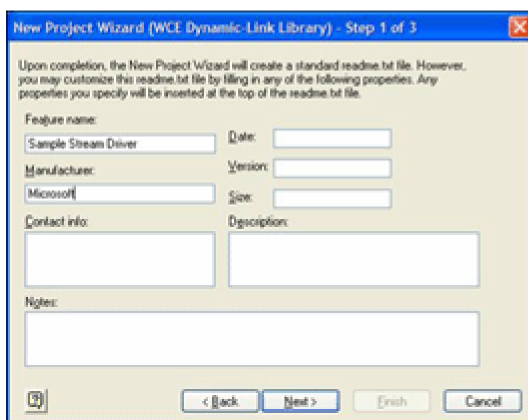
1.用 Platform Builder 打开现有的 **MyPlatform** 工作区。

2.在 **File** 菜单上，单击 **New Project or File**。

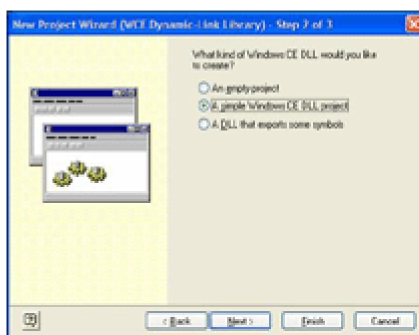
3.选择 **WCE Dynamic-Link Library**，给它一个合适的名称（例如，**StreamDrv**），然后单击 **OK**，如下图所示。



4.在下图所显示的页面中多少填写一些您需要的信息，然后单击 **Next**。



5. 单击 **A simple Windows CE DLL project**，如下图所示。



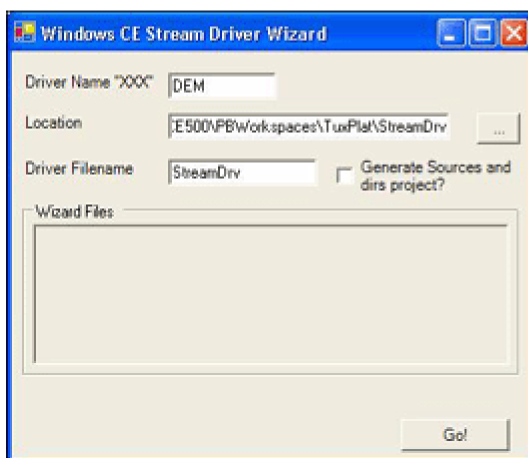
6. 单击 **Finish** 完成此向导。

此时，DLL 只包含一个空的 **DllMain** 函数。您可以公开一些应用程序要调用的函数，并公开一些资源（可能使之成为识别语言/文化的应用程序的一部分），或者使之成为一个设备驱动程序。在本文中，您将使用 Windows CE Stream Driver Wizard 创建您的主干流驱动程序。

在 Windows CE 中，打开流驱动程序就像打开文件一样，只需根据唯一的三字母前缀（例如，COM）。

7. 为您的驱动程序选择一个唯一的三字母标识符。在 **Location** 框中输入您之前创建的流驱动程序的完整路径。或者使用“browse”按钮定位到 Platform Builder 安装中的 PBWorkspaces 目录，找到您前面创建的平台，然后找到流驱动程序的名称（在前面的示例中，此路径为 PBWorkspaces\TuxPlat\StreamDrv）。

8. 在 **Driver Filename** 框中输入驱动程序的名称。如下图所示，使用与您前面使用名称（**StreamDrv**）相同的名称，以确保改写在 Platform Builder 中创建的原始文件。



9. 按 **Go**，将生成流驱动程序源代码。

第二部分：测试流驱动程序测试代码

现在您已经编写了用于 Windows CE 的自定义流驱动程序的基本代码。此时，驱动程序还没有与任何硬件连接。

在 编写完驱动程序之后，您需要为开发人员提供一种测试它的方法。Windows CE 附带了 Windows CE Test Kit (CETK)，它提供了用于各种驱动程序类型的驱动程序测试，包含网络连接、蓝牙、串行端口以及显示。您编写的驱动程序是一种自定义的流驱动程序，它没有 公开与现有的驱动程序测试一样的功能，因此您需要为该驱动程序编写一个自定义测试。虽然您完全可以编写一个应用程序来演练驱动程序，但提供一个 CETK 模块或许更好些，在开发期间可以使用此模块，并且还可以将此模块提供给客户，供他们在装配硬件上测试驱动程序。

在这一部分的练习中，您将执行以下过程：

•创建主干 Tux 模块

•将自定义驱动程序的测试代码添加到 Tux DLL 中

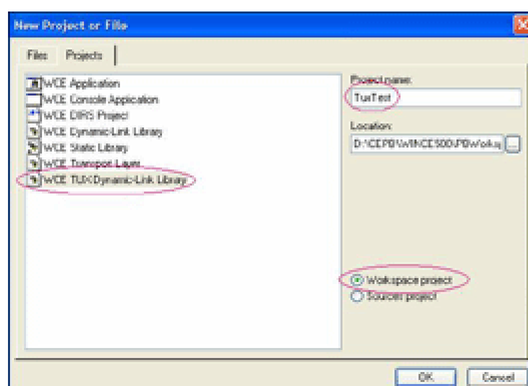
•重新构建操作系统

•设置断点

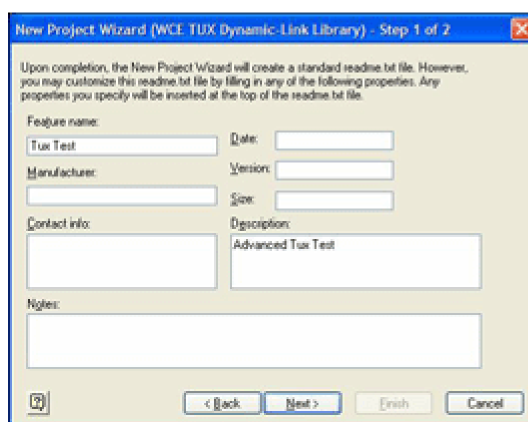
创建主干 Tux 模块

1. 在 Platform Builder 中，在 **File** 菜单上单击 **New Project or File**。

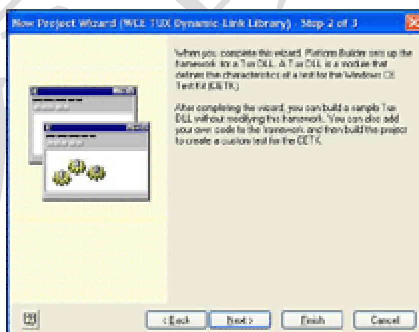
2. 选择 **WCE TUX Dynamic-Link Library**，键入 **TuxTest** 作为项目名称，输入一个位置，单击 **Workspace Project**，然后单击 **OK**，如下图所示。（实际上，您可以选择任意一个项目类型；对于本文，单击 **Workspace Project**）。



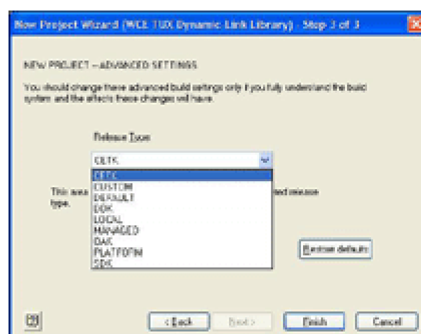
3. 在下图显示的页面中多少填写一些您需要的信息，然后单击 **Next**。



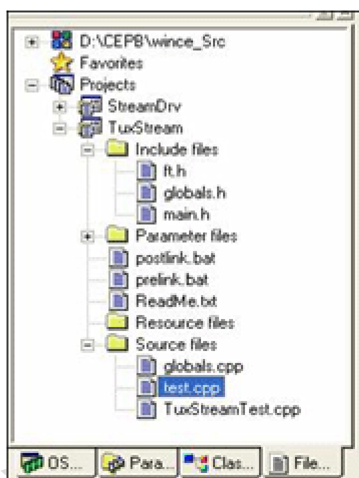
4. 阅读下图所显示的屏幕上的信息，然后单击 **Next**。



5. 在最后一页上，您可以选择选取 **Release Type** 下的 **CETK**，如下图所示。该选项关闭了某些二进制的优化，以提高调试工作效率。单击 **Finish**。



6.单击 **View** | **File View**，然后展开 **Projects** 树显示 tux 源代码，如下图所示。



前图中需要注意的重要文件是：

•ft.h — 该文件包含 tux DLL 所用的函数表。

•test.cpp — 该文件包含从该函数表中调用的测试过程。

TuxStreamTest.cpp — 该文件包含 **DLLMain** 和 **ShellProc**，后者是从 Tux.exe 调用的。

将自定义驱动程序测试代码添加到 Tux DLL 中

1.打开源代码 Test.cpp。

2.使用 CodeClip 来获得 **Tux_Custom_Test** | **TuxCode** 源代码。

3.用 CodeClip 中的代码替代函数 **TestProc** 中的内容。

您会注意到，Test.cpp 中的代码加载了一个名为 Demo.dll 的驱动程序。对于本文，您创建了一个名为 StreamDrv 的驱动程序。您需要修改源代码以加载您的 StreamDrv.dll 驱动程序。

4.找到 Test.cpp 中调用 **LoadLibrary** 的源代码的位置，然后将要从 **Demo.dll** 中加载的驱动程序的名称修改为 **StreamDrv.dll**。

5. 在 Platform Builder 文件视图中，右键单击 **TuxTest** 项目，然后单击 **Build Current Project**。

您还需要从该目录中添加 Windows CE Test Kit 组件。

6. 在 **Device Drivers** 下，找到该目录中 **Windows CE Test Kit** 组件的位置，然后选择 **Add the Windows CE Test Kit**，将该组件添加到您的平台中。

注 将该组件添加到您的平台上并没有将任何文件添加到最后的操作系统映像中；它将 Clientside 文件添加到 build release 文件夹中。您可以从 Platform Builder 下载 Clientside 应用程序，并在目标设备上运行该应用程序。

现在您需要重新构建您的操作系统，以便合并这些变更。

重新构建操作系统

•在 Platform Builder 中，选择 **BuildOS** | **Sysgen**。

构建过程将会花大约 5 分钟完成。

当加载驱动程序时，在流驱动程序的入口点设置一个断点来观察非常有用。

设置断点

1. 单击 **File View**，打开 **StreamDrv** 项目，然后打开 **Source files**。

2. 找到并打开 StreamDrv.cpp。

3. 找到 **DllMain**，然后找到并单击 **switch** 语句。

4. 按 F9 设置断点。

5. 单击 **Target** | **Attach**，将操作系统下载到模拟环境中。

您会看到以下调试输出，断点将启用。注意，在加载操作系统的用户接口（UI）之前，这早就发生了。

```
4294780036 PID:23f767b6 TID:23f767e6 0x83fa6800: >>> Loading module streamdrv.dll at
address 0x01ED0000-0x01ED5000
Loaded symbols for
'C:\WINCE500\PBWORKSPACES\DRVDEMO\RELDIR\EMULATOR_X86_DEBUG\STREAMDRV.DLL'
```

6. 单击 **switch** 语句，然后按 F9 禁用断点。

7. 按 F5，允许操作系统继续加载。

现在，您已经构建了一个 Windows CE 5.0 操作系统，它包含一个自定义流驱动程序，并且您已经在操作系统引导顺序的过程中看到了驱动程序加载。

第三部分：检验驱动程序

在这一部分的练习中，您将执行以下过程：

•使用命令行工具查看从驱动程序公开的函数

•使用远程系统信息（Remote System Information）工具检验驱动程序

•确定驱动程序已加载

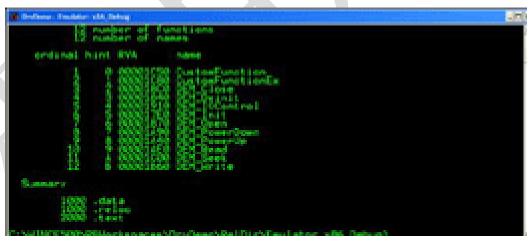
检验您所创建的设备驱动程序的第一种方法是查看从该驱动程序公开的函数。Windows CE 附带了一个名为 Dumpbin 的命令行工具，可以用于检验导入应用程序或模块的内容，或者从 DLL（或驱动程序）导出的内容。

使用命令行工具查看从驱动程序公开的函数

1. 在 Platform Builder 中，单击 **Build OS** | **Open Release Directory**。该操作为当前的工作区打开 build release 文件夹中的 Command Prompt 窗口。

2. 键入 **dumpbin exports StreamDrv.dll**

   下图显示输出。您可以看到，所有需要的流驱动程序函数都是从驱动程序公开的；函数是从 DLL 公开的（通过该项目的 .def 文件）。



3. 键入 **Exit** 关闭 Command Prompt 窗口

   StreamDrv.def 文件的内容如下所示。

```
LIBRARY DemoDriver
EXPORTS
   DEM_Init
   DEM_Deinit
   DEM_Open
   DEM_Close
   DEM_IOControl
   DEM_PowerUp
   DEM_PowerDown
   DEM_Read
   DEM_Write
```

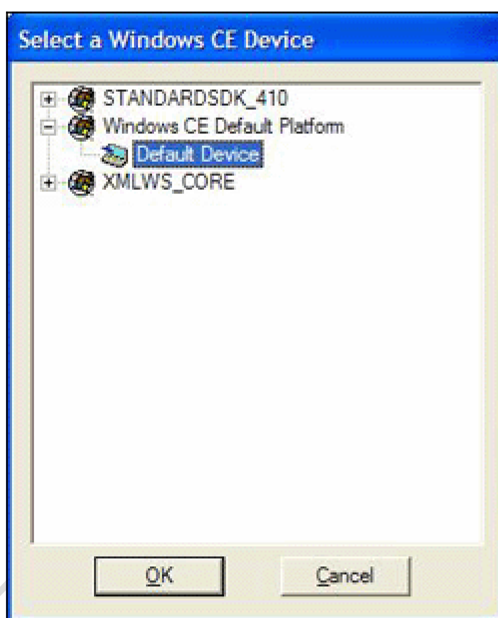
DEM_Seek
CustomFunction
CustomFunctionEx

您可以检验驱动程序的第二种方法是通过远程系统信息工具。

通过远程系统信息工具检验驱动程序

1.在 Platform Builder 中，单击 **Tools** | **Remote System Information**。

2.选择 **Windows CE Default Platform** | **Default Device**，然后单击 **OK**，如下图所示。



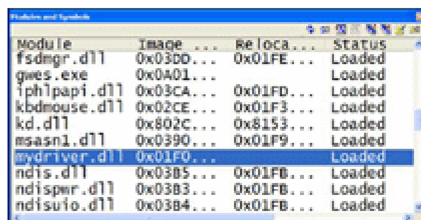
此过程将远程系统信息应用程序连接到 Platform Builder 正在使用的当前活动平台上。下图显示了结果。

您也可以使用加载模块列表来确定已加载了您的驱动程序。

确定驱动程序已加载

•在 Platform Builder 中，使用 Target Control 窗口 (**gi mod**) 或 **View** | **Debug Windows** | **Modules and Symbols**。

下图显示了此过程的结果。



第四部分：使用 Windows CE Test Kit

Windows CE Test Kit 包含设备端组件和桌面组件。设备端组件叫做 Clientside.exe，通过从目录中添加 CETK 组件，您可以将设备端组件添加到您的工作区中。注意，将 Clientside.exe 应用程序添加到工作区中并没有将任何文件添加到最终操作系统映像中，但它却将应用程序复制到 build release 文件夹中。

在桌面计算机上运行 CETK 之前，您需要启动设备上的 Clientside.exe 应用程序。没有链接工具（比如远程工具）的原因在于，CETK 也将运行在装配（零售）设备（比如 Pocket PC）上。

在这一部分的练习中，您将执行以下过程：

1.检验 Windows CE Test Kit 用户接口

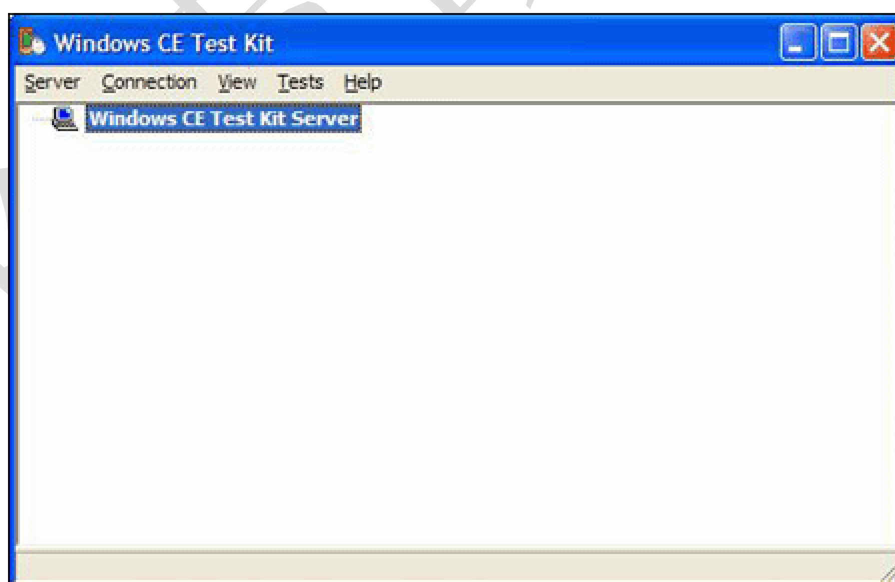
2.运行一个标准测试

检验 Windows CE Test Kit 用户接口

•在 Platform Builder 中，在 **Tools** 菜单上单击 **Windows CE Test Kit**。

这 一步启动 Windows CE Test Kit 应用程序，如下图所示。注意，这不是一个标准的远程工具。Windows CE 附带的大多数远程工具都使用 Kernel Independent Transport Layer (KITL)，一种将工具从基础通信硬件中抽象出来的传输，以便这些工具可以运行在以太网、串行端口、1394、USB 或者其他传输上。

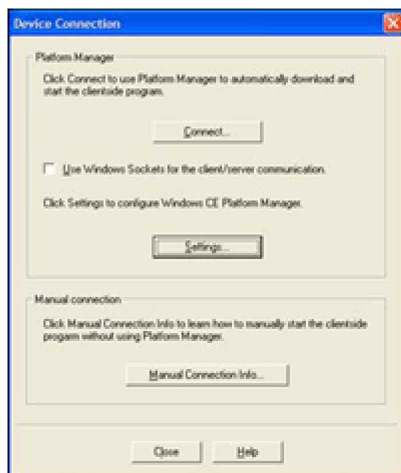
虽然对于 Windows CE 5.0，Windows CE Test Kit 通常通过套接字连接，但是也已经更新了工具来支持 KITL。



•在 Windows CE Test Kit 中，单击 **Connection** | **Start Client**。

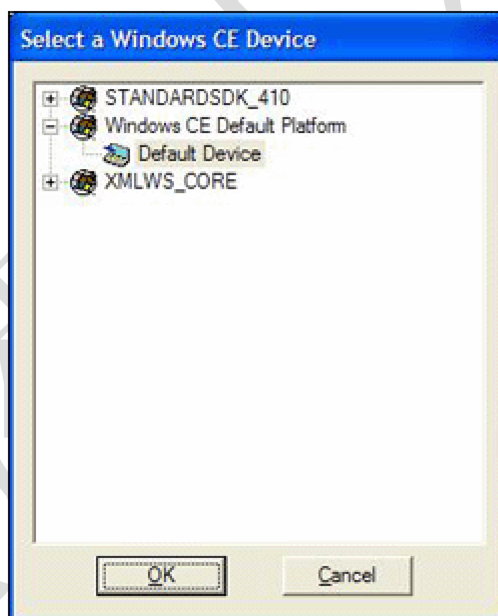
这一步显示 **Device Connection** 对话框，其中您可以选择是通过套接字连接还是通过 KITL 连接。

•确保清除了 **Use Windows Sockets for the client/server communication** 复选框，如下图所示。



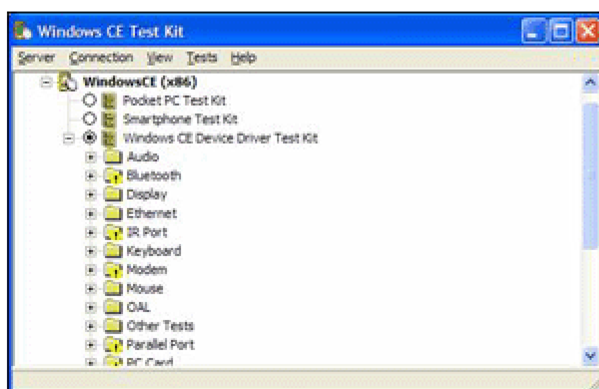
•单击 **Connect**。

在远程工具（KITL）的标准用户界面中，选择 **Windows CE Default Platform** | **Default Device**，然后单击 **OK**，如下图所示。



该过程在目标设备上启动 Clientside.exe，并连接到目标设备上。在完成连接之后，CETK 枚举目标平台上支持的设备，并禁用 CETK 中不支持的设备。

在 CETK 连接到目标设备并枚举设备之后，UI 如下图所示。注意，禁用了某些硬件类别，比如 **Bluetooth IR Port** 和 **Modem**。



将自定义测试添加到 CETK 中之前，您可以运行一个标准测试，以查看测试工作如何进行。

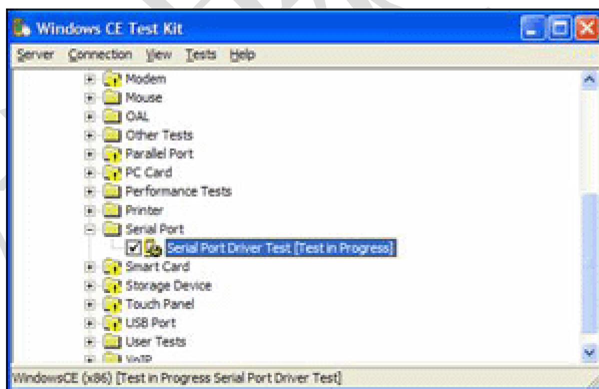
运行标准测试

1.在 CETK 中，展开 **Windows CE (x86)**。

2.找到并展开 **Serial Port**。

3.右键单击 **Serial Port Driver Test**，然后单击 **Quick Start**。

这一步只运行了这一个测试，还没有运行所选的其他测试。UI 指示测试正在进行，如下图所示。

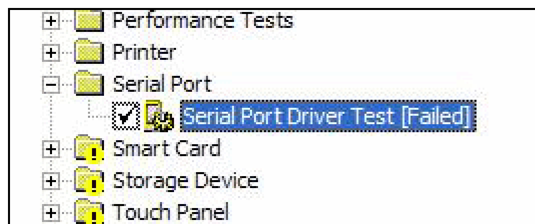


CETK 提供测试过程和测试输出的更新。您也可以在 Platform Builder 中检验调试输出，以便查看测试过程，如下例所示。

```
405910 PID:83d4ee4a TID:83ea5a8a *** Test Name:      Set event mask and wait for thread to
close comm port handle
 405920 PID:83d4ee4a TID:83ea5a8a *** Test ID:         1007
 405920 PID:83d4ee4a TID:83ea5a8a *** Library Path:    \serdrvbvt.dll
 405920 PID:83d4ee4a TID:83ea5a8a *** Command Line:
 405920 PID:83d4ee4a TID:83ea5a8a *** Result:          Passed
 405920 PID:83d4ee4a TID:83ea5a8a *** Random Seed:     15595
 405930 PID:83d4ee4a TID:83ea5a8a *** Thread Count:    1
 405930 PID:83d4ee4a TID:83ea5a8a *** Execution Time: 0:00:05.110
```

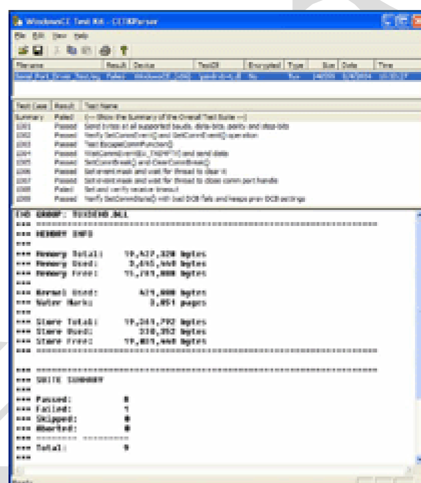
405930 PID:83d4ee4a TID:83ea5a8a ***
^^^

如果 CETK UI 指示模拟器上的串行端口测试已经失败（如下图所示），那么失败可能不是由于每个测试的完全失败而导致的。它可能表明，全部测试套件只有一部分已经失败，并且这部分实际上也是期望的行为。



•右键单击 **Serial Port Driver Test [Failed]**，然后单击 **View Results**。

出现如下图所示的窗口。



查看上图所示的结果，您可以看到，已经运行了 10 个单独的测试。除了 **Set and verify receive timeout** 以外，所有这些测试都已经通过。

要获得更多信息，您可以单击个别测试。

第五部分：创建自定义 CETK 测试

通过使用 Platform Builder User-Defined Test Wizard，您可以创建一个自定义 CETK 测试。该测试将验证自定义流驱动程序（您也已经将其添加到平台中）的导出函数。

在这一部分的练习中，您将执行以下过程：

•列出 CETK 中的自定义流驱动程序测试

•运行自定义流驱动程序测试

列出 CETK 中的自定义流驱动程序测试

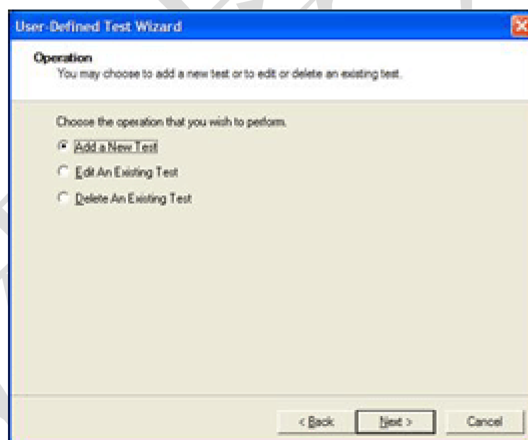
1. 在 CETK 中，单击 **Tests** | **User Defined**。

   这一步启动 User-Defined Test Wizard。该向导的第一页只是一些信息。

2. 单击 **Next**，如下图所示。



3. 单击 **Add a New Test**，然后单击 **Next**，如下图所示。



4. 输入下列信息，然后单击 **Next**：

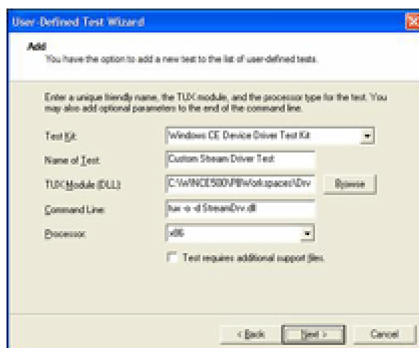
   1. 在 **Name of Test** 框中键入 **Custom Stream Driver Test**

   2. 在 **Tux Module (DLL)** 框中，定位到 C:\Wince500\PBWorkspaces\MyPlatform\RelDir\Emulator_x86_Debug 目录，然后选择 **test.dll** 或 **TuxTest.dll**（这依赖于您在 Platform Builder 中所使用的 Tux 测试的名称）。

   3. 在 **Command Line** 框中，保留当前测试的默认设置。

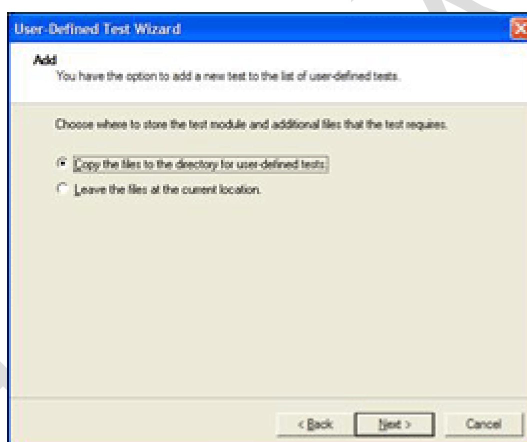
   4. 在 **Processor** 框中键入 **x86**

下图显示信息如何出现在当前的向导页中。

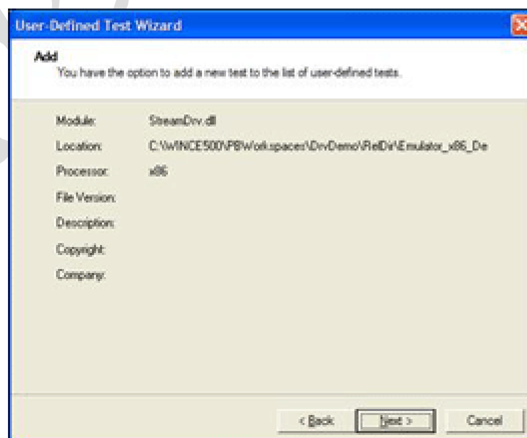


5.单击 **Copy the files to the directory for user-defined tests**，然后单击 **Next**，如下图所示。

您需要将自定义驱动程序测试（您的 DLL）复制到用户定义的测试文件夹中。如果您要删除现有的工作区，那么自定义驱动程序测试仍然保持完好。



1.单击 **Next**，如下图所示。



2.单击 **Finish**，如下图所示。

CETK 应用程序不会用新的测试进行自动刷新。您需要重新同步桌面应用程序，以查看新添加的测试。



3. 右键单击 **Windows CE (x86)**，然后单击 **Redetect Peripherals**。

该过程添加了一个名为 **User Tests** 的新驱动程序类别。您只添加了一个测试，因此，当您展开这个项目时，您只能看到 **Custom Stream Driver Test**。

注 已经将自定义流驱动程序测试的 DLL 复制到下列位置： C:\Program Files\Windows CE Platform Builder\5.00\CEPB\wcetk\user\x86.

运行自定义流驱动程序测试

1. 在可用的测试列表中展开 **User Tests**。

2. 右键单击 **Custom Stream Driver Test**，然后单击 **Quick Start**。

注意 Platform Builder 中显示的下列调试信息。

```
1162630 PID:3c92032 TID:3efe3ea *** TEST STARTING
1162630 PID:3c92032 TID:3efe3ea ***
1162630 PID:3c92032 TID:3efe3ea *** Test Name:      Sample test
1162630 PID:3c92032 TID:3efe3ea *** Test ID:        1
1162640 PID:3c92032 TID:3efe3ea *** Library Path:   \test.dll
1162650 PID:3c92032 TID:3efe3ea *** Command Line:
1162650 PID:3c92032 TID:3efe3ea *** Random Seed:    26648
1162650 PID:3c92032 TID:3efe3ea *** Thread Count:   0
1162650 PID:3c92032 TID:3efe3ea ***
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
1162660 PID:3c92032 TID:3efe3ea test: ShellProc(SPM_BEGIN_TEST, ...) called
1162660 PID:3c92032 TID:3efe3ea    BEGIN TEST: "Sample test", Threads=0, Seed=26648
1162690 PID:3c92032 TID:3efe3ea       Custom Stream Driver Test Starting
1162690 PID:3c92032 TID:3efe3ea       Custom Driver Test - Loading Demo.DLL
1162710 PID:3c92032 TID:3efe3ea 0x83d3dc28: >>> Loading module streamdrv.dll at address
0x01ED0000-0x01ED5000
1162720 PID:3c92032 TID:3efe3ea StreamDrv - DLL_PROCESS_ATTACH
1162720 PID:3c92032 TID:3efe3ea       Custom Driver Test - Loaded Demo.DLL OK
```

```
1162740 PID:3c92032 TID:3efe3ea       Custom Driver Test - Checking Interfaces...
1162740 PID:3c92032 TID:3efe3ea       Custom Driver Test - Checking DEM_Open
1162740 PID:3c92032 TID:3efe3ea       Custom Driver Test - DEM_Open OK
1162740 PID:3c92032 TID:3efe3ea       Custom Driver Test - Checking DEM_Close
1162750 PID:3c92032 TID:3efe3ea       Custom Driver Test - DEM_Close OK
1162750 PID:3c92032 TID:3efe3ea       Custom Driver Test - Checking DEM_Read
1162750 PID:3c92032 TID:3efe3ea       Custom Driver Test - DEM_Read OK
1162770 PID:3c92032 TID:3efe3ea       Custom Driver Test - Checking DEM_Write
1162790 PID:3c92032 TID:3efe3ea       Custom Driver Test - DEM_Write OK
1162790 PID:3c92032 TID:3efe3ea       Custom Driver Test - Checking DEM_Init
1162790 PID:3c92032 TID:3efe3ea       Custom Driver Test - DEM_Init OK
1162790 PID:3c92032 TID:3efe3ea       Custom Driver Test - Checking DEM_Deinit
1162800 PID:3c92032 TID:3efe3ea       Custom Driver Test - DEM_Deinit OK
1162800 PID:3c92032 TID:3efe3ea       Custom Driver Test - Checking DEM_PowerUp
1162800 PID:3c92032 TID:3efe3ea       Custom Driver Test - DEM_PowerUp OK
1162800 PID:3c92032 TID:3efe3ea       Custom Driver Test - Checking DEM_PowerDown
1162810 PID:3c92032 TID:3efe3ea       Custom Driver Test - DEM_PowerDown OK
1162810 PID:3c92032 TID:3efe3ea       Custom Driver Test - Checking DEM_IOControl
1162810 PID:3c92032 TID:3efe3ea       Custom Driver Test - DEM_IOControl OK
1162810 PID:3c92032 TID:3efe3ea       Custom Driver Test - Checking DEM_Seek
1162820 PID:3c92032 TID:3efe3ea       Custom Driver Test - DEM_Seek OK
1162830 PID:3c92032 TID:3efe3ea StreamDrv - DLL_PROCESS_DETACH
1162840 PID:3c92032 TID:3efe3ea 0x83d3dc28: <<< Unloading module streamdrv.dll at address
0x01ED0000-0x01ED5000
1162870 PID:3c92032 TID:3efe3ea test: ShellProc(SPM_END_TEST, ...) called
1162870 PID:3c92032 TID:3efe3ea    END TEST: "Sample test", PASSED, Time=0.180
1162870 PID:3c92032 TID:3efe3ea ***
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

1162870 PID:3c92032 TID:3efe3ea *** TEST COMPLETED
1162880 PID:3c92032 TID:3efe3ea ***
1162880 PID:3c92032 TID:3efe3ea *** Test Name:      Sample test
1162880 PID:3c92032 TID:3efe3ea *** Test ID:        1
1162890 PID:3c92032 TID:3efe3ea *** Library Path:   \test.dll
1162890 PID:3c92032 TID:3efe3ea *** Command Line:
1162890 PID:3c92032 TID:3efe3ea *** Result:         Passed
1162900 PID:3c92032 TID:3efe3ea *** Random Seed:    26648
1162910 PID:3c92032 TID:3efe3ea *** Thread Count:   1
1162910 PID:3c92032 TID:3efe3ea *** Execution Time: 0:00:00.180
```

测试完成，没有任何警告或错误。您也可以在客户端检验测试结果。

第六部分：确定谁拥有流驱动程序

到目前为止，您已经通过 Platform Builder 调试信息、驱动程序源代码中的断点以及自定义 CETK 测试看到了自定义流驱动程序加载。

在这一部分的练习中，您将执行以下过程：

•使用远程进程查看器（Remote Process Viewer）确定哪个进程正在加载驱动程序

•显示信息

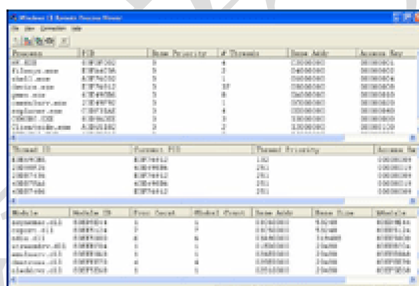
使用远程进程查看器确定哪个进程正在加载驱动程序

1. 在 Platform Builder 中，单击 **Tool** | **Remote Process Viewer**。

除了加载到每个进程地址空间中的 DLL 或模块，进程查看器应用程序会显示当前正在运行的进程列表。

2. 找到并选择 **device.exe**。

下图显示加载到 Device.exe 进程地址空间中的 DLL 列表。Device.exe 是 Windows CE 的设备驱动程序管理器。



显示信息

•通过使用 Platform Builder，编写一个 Microsoft Win32 应用程序来将数据写入自定义流驱动程序中，并读回和显示该信息。您可以使用 MessageBox API 来完成。

您需要用于该任务的 API 是：

•CreateFile

•WriteFile

•ReadFile

•CloseHandle

•MessageBox

小结

•驱动程序只不过是一些 DLL。

•驱动程序被映射到 Device.exe 进程地址空间中。

•驱动程序公开了一个众所周知的接口集。

•将设备驱动程序添加到 Windows CE 操作系统映像中相对容易的多。

•可以使用 Windows CE Test Kit 来协助进行平台开发和调试。

•编写自定义 CETK 测试相当容易。

附录五 Windows CE .NET 中的文件系统体系结构

Windows CE .NET 文件系统是一种灵活的模块化设计，它允许自定义文件系统、筛选器和多种不同的块设备类型。文件系统和所有与文件相关的 API 都是通过 FileSys.exe 进程来管理的。这个模块实现了对象存储和存储管理器（我们将稍微讨论一下对象存储），并将所有文件系统统一到一个根"\"下面的单个系统中。在 Windows CE .NET 中，所有文件和文件系统都存在于从"\"作为根开始的单个命名空间中。所有文件均以在层次结构树中从根开始的唯一路径进行标识。这类似于桌面计算机版本的 Windows，只是没有驱动器号。在 Windows CE 中，驱动器作为文件夹装入根的下面。因此，添加到系统中的新存储卡将装入树的根中，其路径类似于"\Storage Card"。

FileSys.exe 由下列几个组件组成：

- ROM 文件系统
- 存储管理器
- 对象存储

对象存储是一个内存堆，由 FileSys.exe 控制。对象存储包含 RAM 系统注册表、RAM 文件系统和属性数据库。它们都是 FileSys.exe 模块的可选组件。RAM 文件系统和属性数据库是完全可选的，并且在某些系统中可以根本不存在。对每个 Windows CE 设备来说，以某些形式存在的注册表是必需的。使用 Windows CE .NET，它可以作为文件存在于外部装入的文件系统（例如磁盘）中。随后，我们将了解注册表和文件系统是如何产生联系的。

基于 RAM 的文件系统通常连接到呈现给应用程序的统一文件系统的根。就是说，文件"\MyFile.txt"位于统一系统的根和 RAM 文件系统的根中。ROM 文件系统连接到统一文件系统中的"\Windows"文件夹。这意味着，ROM 中的 ***所有*** 文件均可作为"\Windows"文件夹中的只读文件来访问。

存储管理器 (Storage Manager) 是 Windows CE .NET 的新功能。如名称所示，它负责管理系统中的存储设备，以及用于访问它们的文件系统。存储管理器处理 4 种主要项目：

- **存储驱动程序。**它们是物理存储介质的设备驱动程序。它们有时称为"块驱动程序"，这是因为它们提供对数据存储的随机寻址块的访问。
- **分区驱动程序。**它们为单个存储设备上的多个分区提供管理。Windows CE .NET 允许物理磁盘包含多个分区，并且每个分区可以格式化为不同的文件系统。分区驱动程序实际上是存储驱动

程序的转换器。它公开与存储驱动程序相同的接口，并将分区的块地址转换为存储设备块的真实地址。然后，它将调用传递给存储驱动程序。

- **文件系统驱动程序。**这些驱动程序将存储设备上的数据组织为文件和文件夹。Windows CE .NET 附带了几个不同的系统，包括用于 CD 和 DVD 的 UDFS，以及 FATFS（包括 FAT32 支持）。在 4.2 版本中，有一个新的系统，称为事务安全 FAT 文件系统 (TFAT)。（我们可能在以后的文章中讨论它；同样，如果您对此有兴趣，请告诉我们。）
- **文件系统筛选器。**文件系统筛选器用于处理对文件系统的调用，此后，文件系统才能获得这些调用。这就允许对文件访问进行某些特殊的处理，以便进行数据加密、压缩和使用统计数据进行监视。

正如谚语所说，百闻不如一见，下图说明了文件系统的各个组件之间的关系。

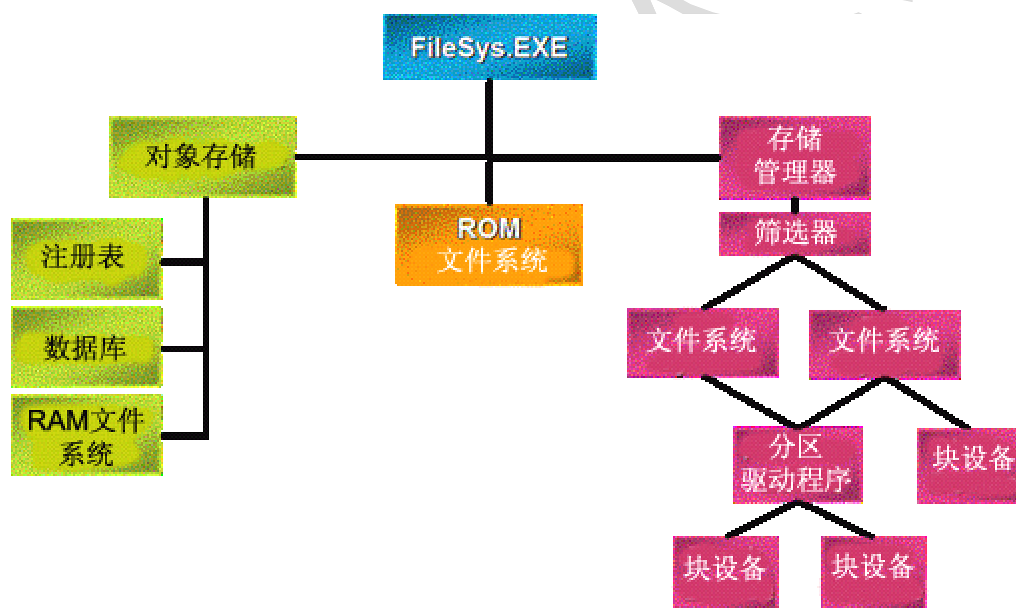


图 1. Windows CE 文件系统概述

有关该结构，一个要注意到的重要事情是文件系统筛选器工作于存储管理器的下面，并且无法应用于对象存储中的 ROM 文件系统或 RAM 文件系统。此时，Microsoft 没有为筛选对这些系统的访问提供机制。因此，在本文中我们将重点讨论上图右侧的内容。为了让您看得更清楚，下图放大了这片区域。

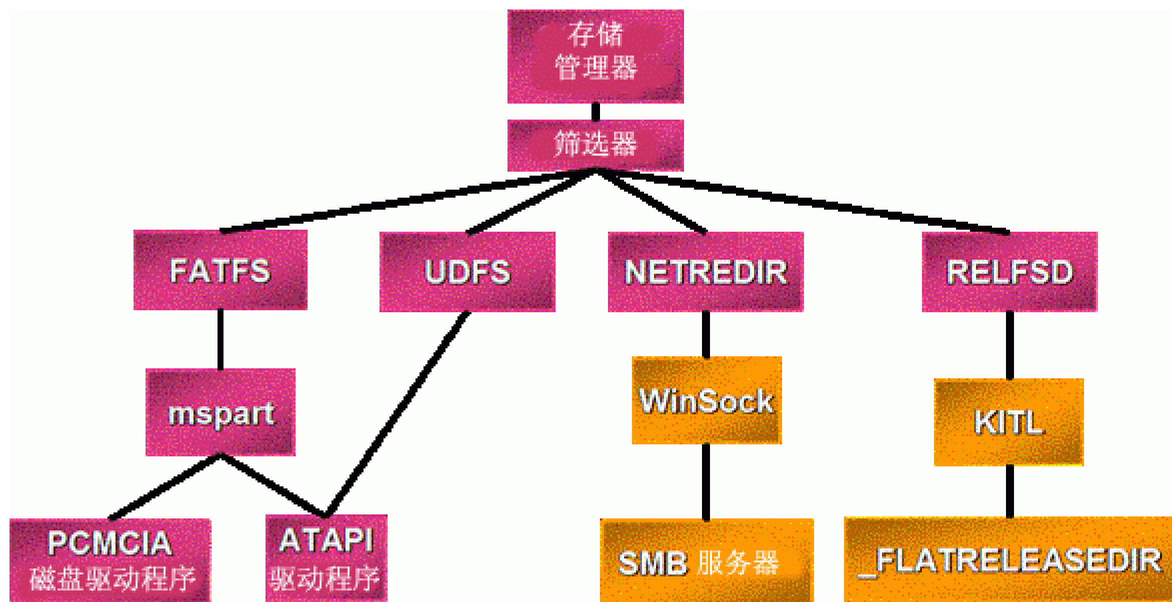


图 2. 存储管理器和相关组件

在上图中可以看见，并非所有文件系统驱动程序都使用了物理设备，即使使用，也可能没有使用分区驱动程序。这就提供了巨大的灵活性。例如，负责提供网络共享访问的网络重定向器使用 WinSock 通过网络与远程服务器通信，并且它在 Windows CE 设备上没有物理磁盘。

既然我们可以看见大多数项目是什么，以及它们是如何相互关联的，下面我们将讨论系统如何加载所有各项。操作系统启动时，NK.exe 将直接从 ROM 文件系统加载 FileSys.exe。然后，FileSys.exe 从 ROM 文件系统内的默认注册表对注册表进行初始化。（这里，在使用配置单元注册表时有一个类似"先有鸡还是先有蛋"的问题，这是因为注册表在磁盘上的文件中，而文件系统还没有装入。在随后的文章介绍配置单元注册表时，我们将介绍操作系统是如何解决这个问题的。）

然后，FileSys.exe 将读取注册表项，以便启动各种应用程序。首先列在注册表中的一个应用程序通常是 Device.exe，即设备管理器。设备管理器从 HKEY_LOCAL_MACHINE\Driver\BuiltIn 项加载驱动程序。正常情况下，任何内置的磁盘设备（例如，硬盘）列在该项下面，所以将加载块驱动程序。块驱动程序通告一个特定的设备类标识符 BLOCK_DRIVER_GUID {A4E7EDDA-E575-4252-9D6B-4195D48BB865}。

内置到 FileSys.exe 中的存储管理器向设备管理器通知系统注册，以便接收有关块驱动程序加载和卸载的通知。然后，存储管理器打开块驱动程序，并向它查询配置文件名称。每个块设备类型都有一个与它相关的配置文件。PROFILE 是一个注册表项，用于指定特定类型设备的分区驱动程序和默认文件系统。（我们将对配置文件的注册表项的细节稍加介绍。）

存储管理器读取有关设备的分区驱动程序的信息，并加载适当的驱动程序。（Microsoft 提供了一个称为"mspar"的分区驱动程序，用于通过磁盘的主启动记录中的分区表进行标准硬盘分区。当然，如果需要，您可以随便创建您自己的分区，也可以根本不使用它。）

一旦分区驱动程序已加载，然后存储管理器将请求分区驱动程序枚举磁盘上的分区，并标识每个分区上的文件系统。分区驱动程序将从主启动记录（MBR）中读取有关分区和文件系统的信息，并向存储管理器提供信息，然后，存储管理器使用该信息来加载每个分区的文件系统驱动程序，并将文件系统装入到统一文件系统的根中。虽然这似乎有很多步骤，但它允许完全在同一个框架中灵活地支持网络重定向器 FATFS 和 DVD ROM。

在了解 FileSys.exe 如何加载各种组件的步骤以后，我们将更为详细地介绍文件系统驱动程序和文件系统驱动程序管理器（FSDMGR）的角色。FSDMGR 是存储管理器（在该操作系统以前的版本中它是设备管理器的一部分）的一部分，负责向文件系统驱动程序提供服务。因为文件系统将不需要知道数据是否来自磁盘上的分区、或者直接来自磁盘，所以，FSDMGR 对文件系统驱动程序进行包装，以便为驱动程序的高端或低端提供接口。下图说明这是如何工作的。



图 3. 存储管理器

存储管理器调用文件系统驱动程序 (FSD)，而 FSD 使用 FSDMGR_ API 从设备检索数据。如果是 CD（没有分区），则设备通过 FSDMGR 与块驱动程序通信。如果它是有多个分区的硬盘，那么它以同样方式使用 FSDMGR_ API。但这之后 FSDMGR 会将工作转交给适当的分区驱动程序。

我们已经讨论了存储管理器、FSDMGR、FSD、分区驱动程序和块驱动程序如何交互和互操作。让我们回过来详细讨论它们是如何加载的，并考查注册表中的配置文件的细节。前面已经提到过，配置文件只是一组注册表值，用于定义有关块设备和应当如何在系统中使用它的信息。配置文件位于以下项的下面：
HKEY_LOCAL_MACHINE\System\StorageManager\Profiles

每个配置文件都是位于基本配置文件项的下面，以此配置文件名称标识的项。例如，如果 Windows CE .NET 设备上有一个硬盘，并且它确实使用硬盘配置文件，则配置文件位于

HKEY_LOCAL_MACHINE\System\StorageManager\Profiles\Hard Disk 下面。所有配置文件信息都包含在该配置文件项下面的命名值中。下表列出了各种值及其目的。

表 1. 配置文件注册表项

值	类型	说明
Folder	REG_SZ	在 Windows 资源管理器中显示给用户的文件夹名称。对于多个实例，将自动追加整数。（例如，Storage Card、Storage Card2 等等。）
FileSystem	REG_SZ	用作磁盘的默认文件系统的名称。（如果使用了分区驱动程序，则通常不使用它。）
PartitionDriver	REG_SZ	列出如果默认驱动程序不合适时要使用的分区驱动程序。如果该字符串为空，则不加载任何分区驱动程序。如果该值不出现，则使用默认分区驱动程序。
AutoFormat	REG_DWORD	如果磁盘没有格式化，则自动执行格式化
AutoPart	REG_DWORD	如果磁盘没有分区，则自动将它分区，并且其中一个分区占据最大数量的可用磁盘空间。
AutoMount	REG_DWORD	当存储设备驱动程序加载时，自动装入文件系统。
Name	REG_SZ	显示在控制面板 UI 中的配置文件名称。
MountFlags	REG_DWORD	用于确定如何装入文件系统的标志。（详细信息，请参阅下表。）

需要特别注意 MountFlags 值。它是下表中的值的位掩码。

表 2. MountFlags 注册表项的标志

标志	说明
1	隐藏文件系统。
2	可以包含配置单元注册表。
4	作为文件系统的根（"\"）装入。
8	隐藏 ROM 文件系统。（只与 [4] 一起使用。）

将文件系统标记为隐藏可以防止它被任何标准的文件和文件夹枚举发现。（例如 **FindFirstFile** 等等。）存储管理器独立完成该操作，以便设备驱动程序和应用程序可以检测到是否特定系统正在使用存储管理器。（由于较旧版本的操作系统没有它，所以某些驱动程序可能需要与旧的 LoadFSD(Ex) 机制相兼容，以用于加载文件系统。）虽然无法使用 **FindFirstFile** 来枚举任意的隐藏系统，但如果知道文件系统的名称，则可以在任何会使用文件路径的地方使用它。以下示例显示了如何检测存储管理器是否正在使用某个系统。

```
BOOL IsStorageManagerRunning()
{
   DWORD attr = GetFileAttributes( _T("\\StoreMgr") );
   if( (attr != -1) && (attr & FILE_ATTRIBUTE_TEMPORARY) )
      return TRUE;
   return FALSE;
}
```

MountFlags 的下一位指示文件系统是否可包含基于配置单元的注册表。这使 FileSys.exe 能够管理先前提到的"先有鸡还是先有蛋"的问题。（因为需要注册表才能加载访问可能在磁盘上的注册表配置单元文件所需的组件……）我们将在随后的有关配置单元注册表的文章中介绍如何使用这一位。

后面两位是相关的，当您希望将外部文件系统作为统一文件系统的根装入时，则需要使用它们。可以回想起，通常统一系统的根是 RAM 文件系统。这对电池供电的手持设备很有效，但对有时会关闭的交流电源供电的设备却无效，因为 RAM 内容每次关闭时都会丢失。作为根标志的装入文件系统允许您通过将外部存储作为根进行连接来避免这个问题，因为这样一来，文件 \MyDataFile.TXT 将驻留在外部存储设备的根中。隐藏 ROM 文件系统将隐藏 ROM 文件系统数据文件（但不执行适当的 EXE 和 DLL），以便允许您更新 ROM 中的所有文件。这就允许您在闪存中使用非常小的操作系统映像，而将大多数可执行文件放在磁盘上，只在需要时才加载它们。（现在很像桌面计算机系统。）

如果特定配置文件的任何值都不存在，则存储管理器将使用 HKEY_LOCAL_MACHINE\System\StorageManager\Profiles 项中的默认值。可以重写的默认值位于 COMMON.REG 中。您应当使用您的 PLATFORM.REG 或 PROJECT.REG 进行重写。（记住，不应当改变 COMMON.REG！）下表显示了 COMMON.REG 中的默认值。

值	默认值
Folder	LOC_STORE_DEFAULT_FOLDER（用于设备的 .STR 文件中一个字符串的标识符；在英文内部版本中通常是"Storage Card"。）
FileSystem	FATFS
PartitionDriver	Mspart.dll
AutoFormat	0
AutoPart	0
AutoMount	1
MountFlags	0

小结

Windows CE 文件系统体系结构是灵活的和可扩展的，并且支持：

- 多个块设备。
- 每个块设备支持多个分区。
- 每个分区支持不同文件系统。
- 将外部设备文件系统作为根系统装入。